Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Linear Algebra and its Applications 419 (2006) 125–171
LINEAR ALGEBRA AND ITS APPLICATIONS
www.elsevier.com/locate/laa

Principal minors, Part II: The principal minor assignment problem

Kent Griffin, Michael J. Tsatsomeros *

Mathematics Department, Washington State University, Pullman, WA 99164-3113, USA

Received 26 October 2005; accepted 14 April 2006
Available online 12 June 2006
Submitted by M. Neumann

Abstract

The inverse problem of finding a matrix with prescribed principal minors is considered. A condition that implies a constructive algorithm for solving this problem will always succeed is presented. The algorithm is based on reconstructing matrices from their principal submatrices and Schur complements in a recursive manner. Consequences regarding the overdeterminancy of this inverse problem are examined, leading to a faster (polynomial time) version of the algorithmic construction. Care is given in the MATLAB® implementation of the algorithms regarding numerical stability and accuracy.

AMS classification: 15A29; 93B55; 15A15; 65F40

Keywords: Principal submatrix; Inverse eigenvalue problem; Schur complement

1. Introduction

In this paper, which is a natural continuation of our work in [2], we study the following inverse problem:

[PMAP] Find, if possible, an $n \times n$ matrix A having prescribed principal minors.

Recall that a *principal minor* of A is the determinant of a submatrix of A formed by removing k $(0 \leqslant k \leqslant n-1)$ rows and the corresponding columns of A. We refer to the above inverse problem as the *Principal Minor Assignment Problem* (PMAP).

* Corresponding author.
E-mail addresses: kgriffin@math.wsu.edu (K. Griffin), tsat@math.wsu.edu (M.J. Tsatsomeros).

0024-3795/$ - see front matter
doi:10.1016/j.laa.2006.04.009

Some immediate observations and remarks about PMAP are in order. First, PMAP is equivalent to the inverse eigenvalue problem of finding a matrix with prescribed spectra (and thus characteristic polynomials) for all of its principal submatrices. Second, PMAP is a natural algebraic problem with many potential applications akin to inverse eigenvalue and pole assignment problems that arise in engineering and other mathematical sciences. Third, as an $n \times n$ matrix has $2^n - 1$ principal minors and n^2 entries, PMAP is an overdetermined problem for $n \geqslant 5$. As a consequence, the existence of a solution to PMAP depends on relations among the (principal) minors of the matrix being satisfied. Generally, such relations (e.g., Newton identities for each principal submatrix) are theoretically and computationally hard to verify and fulfill.

Our original motivation comes from an open problem in [3], where PMAP is associated to the existence of GKK matrices with prescribed principal minors (see Section 6). The main goal in this paper is to develop and present a constructive algorithm for PMAP, called PM2MAT. This is achieved under a certain condition which guarantees that the algorithm will succeed. The output of PM2MAT is a matrix with the prescribed principal minors, if one indeed exists. Failure to produce an output under this condition signifies the non-existence of a solution (see Section 3.4). The algorithm is based on a method presented in [2] that computes all the principal minors of a matrix recursively.

Although the implementations in MATLAB® of PM2MAT and related functions of this paper are subject to roundoff errors and loss of precision due to cancellation, the PM2MAT algorithm is capable of solving PMAP, under the condition referred to above, exactly. This could be accomplished using any rational arithmetic system that is capable of exactly performing the four arithmetic operations in addition to taking square roots.

The organization of our paper is as follows:

- Section 2: All notation and terminology can be found here.
- Section 3: The algorithm PM2MAT for PMAP is introduced and the theoretical basis for its functionality developed; a detailed description and analysis of PM2MAT follows (initial and main level loops, handling of zero principal minors, deskewing, field considerations, as well as an operation count and strategy for general use).
- Section 4: Several comprehensive examples are presented.
- Section 5: It is shown that generically[1] not all principal minors are needed in the reverse engineering of a matrix (Lemma 5.1 and Theorem 5.3). As a result, a faster version of PM2MAT is developed: FPM2MAT.
- Section 6: Theoretical consequences and applications of PMAP and PM2MAT are discussed. Also some statements in an old related paper [5] are corrected.
- Section 7: Some possible future work is outlined.
- Appendices A–G: Source code for PM2MAT, PMFRONT, FMAT2PM, FPM2MAT, V2IDX, IDX2V, PMSHOW in MATLAB®[2] are found here. PMFRONT is the front end for generally solving PMAP. All codes (including the ones related to [2] and MAT2PM) are available for download on the Web[3] or via email from the authors.

[1] We refer to a property as generic if it holds true for all choices of variables (matrix entries) except those on a strict algebraic subvariety.

[2] Version 7.0.1.15 ($R14$) or later is required.

[3] http://www.math.wsu.edu/math/faculty/tsat/pm.html.

2. Notation and preliminaries

The following technical notation is used:

- $(A)_{ij}$ or A_{ij} is the (i, j)th entry of the matrix A. Similarly, $v_i = v(i)$ is the ith entry of the vector v.
- $\langle n\rangle = \{1, 2, \dots, n\}$ for every positive integer n.
- The lower case Greek letters α, β, γ are used as index sets. Thus, $\alpha, \beta, \gamma \subseteq \langle n\rangle$, and the elements of α, β, γ are assumed to be in ascending order. The number of elements in α is denoted $|\alpha|$.
- Let $\gamma \subseteq \langle n\rangle$ and $\beta = \{\beta_1, \beta_2, \dots, \beta_k\} \subseteq \langle|\gamma|\rangle$. Define the indexing operation $[\gamma]_\beta$ as
$$[\gamma]_\beta := \{\gamma_{\beta_1}, \gamma_{\beta_2}, \dots, \gamma_{\beta_k}\} \subseteq \gamma.$$
- $A[\alpha, \beta]$ is the submatrix of A whose rows and columns are indexed by $\alpha, \beta \subseteq \langle n\rangle$, respectively. When a row or column index set is empty, the corresponding submatrix is considered vacuous and by convention has determinant equal to 1.
- $A[\alpha] := A[\alpha, \alpha]$, $A(\alpha, \beta] := A[\alpha^c, \beta]$; $A[\alpha, \beta)$, $A(\alpha, \beta)$ and $A(\alpha)$ are defined analogously, where α^c is the complement of α with respect to the set $\langle n\rangle$. When $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_k\}$ is known explicitly (as in the examples), we let $A[\alpha_1, \alpha_2, \dots, \alpha_k] := A[\{\alpha_1, \alpha_2, \dots, \alpha_k\}]$.
- The *Schur complement* of an invertible principal submatrix $A[\alpha]$ in A is
$$A/A[\alpha] = A(\alpha) - A(\alpha, \alpha]\,(A[\alpha])^{-1}A[\alpha, \alpha).$$
- $A \in \mathscr{M}_n(\mathbb{C})$ and $B \in \mathscr{M}_n(\mathbb{C})$ are said to be *diagonally similar* if there exists a non-singular diagonal matrix $D \in \mathscr{M}_n(\mathbb{C})$ such that $A = D^{-1}BD$.
- $A \in \mathscr{M}_n(\mathbb{C})$ and $B \in \mathscr{M}_n(\mathbb{C})$ are said to be *diagonally similar with transpose* if either $A = D^{-1}BD$ or $A = D^{-1}B^{\mathrm{T}}D$. Note that this is simple transposition (without conjugation of the entries) in the case of complex matrices.
- $A \in \mathscr{M}_n(\mathbb{C})$ with $n \geqslant 2$ is said to be *reducible* if there exists a permutation matrix $P \in \mathscr{M}_n(\mathbb{R})$ such that $P^{\mathrm{T}}AP = \begin{bmatrix} B & C \\ 0 & D\end{bmatrix}$ where $B \in \mathscr{M}_{n_1(\mathbb{C})}$, $D \in \mathscr{M}_{n_2}(\mathbb{C})$ with $n = n_1 + n_2$ and $n_1, n_2 \geqslant 1$. If $n = 1$, A is reducible if $A = [0]$. Otherwise, A is said to be *irreducible*.

To enable Definition 2.3 below, we define the following similarity:

Definition 2.1. $A, B \in \mathscr{M}_m(\mathbb{C})$ with $A = [a_{ij}]$, $B = [b_{ij}]$ are *dot similar* (written $A \dot{\sim} B$) if for all $i, j \in \langle m\rangle$ there exists $T = [t_{ij}] \in \mathscr{M}_n(\mathbb{C})$ such that $a_{ij} = t_{ij}b_{ij}$ and $t_{ij}t_{ji} = 1$.

Note that if A and B are diagonally similar, then they are dot similar, but the converse is generally false. If the off-diagonal entries of A are nonzero (which implies that the off-diagonal entries of B are nonzero) then it is easy to see that $A \dot{\sim} B$ implies $A^{\mathrm{T}} \dot{\sim} B$.

The set $\mathscr{S}_A$ defined below is also needed for Definition 2.3. It is implicitly assumed that all Schur complements contained in $\mathscr{S}_A$ are well-defined.

Definition 2.2. Given $A \in \mathscr{M}_n(\mathbb{C})$, $n \geqslant 2$, the set of matrices $\mathscr{S}_A$ is defined as the minimal set of matrices such that:

(1) $\mathscr{S}_A$ contains $A(1)$ and $A/A[1]$ and
(2) if $B \in \mathscr{S}_A$ is an $m \times m$ matrix with $m \geqslant 2$, then $B(1)$ and $B/B[1]$ are also in $\mathscr{S}_A$.

Now, we introduce a matrix class for which PM2MAT is guaranteed to succeed (PM2MAT will, however, succeed for a broader class as we shall see).

Definition 2.3. The matrix $A \in \mathscr{M}_n(\mathbb{C})$, $n \geqslant 2$ is said to be *ODF* (off-diagonal full) if the following three conditions hold:

(a) the off-diagonal entries of all the elements of $\mathscr{S}_A$ are nonzero,
(b) all $B \in \mathscr{S}_A$, where $B \in \mathscr{M}_m(\mathbb{C})$ with $m \geqslant 4$, satisfy the property that for all partitions of $\langle m\rangle$ into subsets α, β with $|\alpha| \geqslant 2$, $|\beta| \geqslant 2$, either $\operatorname{rank}(B[\alpha,\beta]) \geqslant 2$ or $\operatorname{rank}(B[\beta,\alpha]) \geqslant 2$ and
(c) for all $C \in \mathscr{S}_A \cup \{A\}$, where $C \in \mathscr{M}_m(\mathbb{C})$ with $m \geqslant 4$, the pair $L = C(1)$ and $R = C/C[1]$ satisfy the property that if $\operatorname{rank}(L - \widehat{R}) = 1$ and $\hat{R} \dot{\sim} R$, then $R = \hat{R}$.

A few remarks concerning this definition are in order:

1. The set $\mathscr{S}_A$ is the set of all intermediate matrices computed by the MAT2PM algorithm of [2] in finding all the principal minors of a matrix $A \in \mathscr{M}_n(\mathbb{C})$ when no zero pivot is encountered. Thus, the role of $\mathscr{S}_A$ in the inverse process of PM2MAT to be developed here is natural.
2. Generically, (a) and (b) are true, and (c) is a technical condition for a case that generically is not encountered in PM2MAT. Assuming (c) enables the PMAP to be solved by PM2MAT in an amount of time comparable to finding all the principal minors of a matrix using MAT2PM.
3. Condition (b) appears in Theorem 3.1 of Loewy. This theorem will be invoked to guarantee the uniqueness of the intermediate matrices PM2MAT computes up to diagonal similarity with transpose.
4. The conditions imposed by the definition of an ODF matrix are systematically and automatically checked in the implementation of PM2MAT and warnings are issued if the conditions are violated.

Finally, by way of review, we restate two Schur complement lemmas from [2]:

Lemma 2.4 (Determinant Property). *Let $A \in \mathscr{M}_n(\mathbb{C})$, $\alpha \subset \langle n\rangle$, where $A[\alpha]$ is nonsingular, and denote $\gamma = \alpha^c$. If $\beta \subseteq \langle|\alpha^c|\rangle$, then*

$$\det(A[\alpha \cup \beta']) = \det(A[\alpha])\ \det((A/A[\alpha])[\beta]),$$

where

$$\beta' = [\gamma]_\beta = \{\gamma_{\beta_1}, \gamma_{\beta_2}, \ldots, \gamma_{\beta_k}\}.$$

Lemma 2.5 (Quotient Property). *Let $A \in \mathscr{M}_n(\mathbb{C})$, $\alpha \subset \langle n\rangle$. As in the previous lemma, let $\beta \subseteq \langle|\alpha^c|\rangle$ and $\beta' = [\alpha^c]_\beta$. If both $A[\alpha]$ and $A[\alpha \cup \beta']$ are nonsingular, then*

$$(A/A[\alpha \cup \beta']) = (A/A[\alpha])/((A/A[\alpha])[\beta]).$$

3. Solving the PMAP via PM2MAT

3.1. Preliminaries

Under certain conditions, the algorithm MAT2PM developed in [2] to compute principal minors can be deliberately reversed to produce a matrix having a given set of principal minors. This process is implemented in the MATLAB® function PM2MAT of Appendix A.

MAT2PM will not be fully described here, but by way of review, it computes all the principal minors of a matrix $A \in \mathscr{M}_n(\mathbb{C})$ in *levels* by repeatedly taking Schur complements and submatrices of Schur complements. Schematically, the operation of MAT2PM (and also PM2MAT in reverse) is summarized in Fig. 1, where pm represents the array of principal minors in a special order. MAT2PM operates from level 0 to level $n-1$ to produce the 2^n-1 principal minors of A.

PM2MAT reverses the process by first taking the 2^n-1 principal minors in pm and producing the 2^{n-1} matrices in $\mathscr{M}_1(\mathbb{C})$ of level $n-1$. It then proceeds to produce 2^{n-2} matrices in $\mathscr{M}_2(\mathbb{C})$ of level $n-2$, and so forth, until a matrix in $\mathscr{M}_n(\mathbb{C})$ having the given set of principal minors is obtained.

To describe the essential step of PM2MAT, it is sufficient to focus on its last step from level 1 to level 0 in Fig. 1. At level 1 we have already computed $C = A/A[1]$ and $B = A(1)$. As shown using Lemma 2.4 and Lemma 2.5 in [2], we can also compute $C[1] = \frac{pm(3)}{pm(1)}$ and $B[1] = pm(2)$ from the input array pm. Thus to find A, we only need to find suitable $A(1, 1]$ and $A[1, 1)$. This can be done (non-uniquely) via the relation

$$A(1,1]A[1,1) = \frac{A(1) - A/A[1]}{A[1]},$$

provided that the quantity $A(1) - A/A[1]$ at hand has the desired rank of 1. However, this is typically not the case. To remedy this situation, $A/A[1]$ is altered by an appropriate diagonal similarity with transpose, leaving the principal minors invariant and achieving the rank condition. See Section 3.2.3 for the details of this operation.

One of the first questions that naturally arises in this process is as follows. If we compute the principal minors of $A \in \mathscr{M}_n(\mathbb{C})$ with MAT2PM and then find a matrix $B \in \mathscr{M}_n(\mathbb{C})$ having equal corresponding principal minors to A with PM2MAT, what is the relationship between A and B? We immediately observe that A need not be equal to B, since both diagonal similarity and transposition clearly preserve principal minors. The following theorem by Loewy [4] states necessary and sufficient conditions under which diagonal similarity with transpose is precisely the relationship that must exist between A and B.

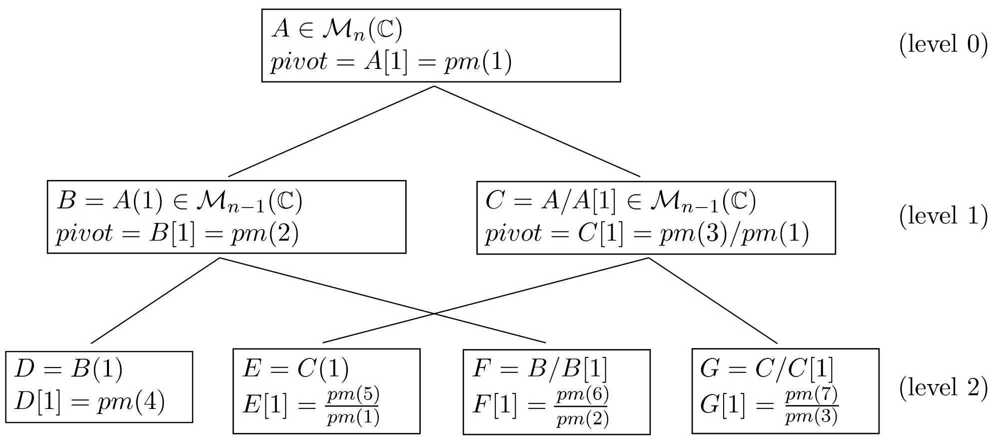

Fig. 1. Three levels of MAT2PM and PM2MAT operation.

Theorem 3.1. *Let $A, B \in \mathscr{M}_n(\mathbb{C})$. Suppose $n \geqslant 4$, A is irreducible, and for every partition of $\langle n \rangle$ into subsets α, β with $|\alpha| \geqslant 2$, $|\beta| \geqslant 2$ either $\operatorname{rank}(A[\alpha, \beta]) \geqslant 2$ or $\operatorname{rank}(A[\beta, \alpha]) \geqslant 2$. Then A and B have equal corresponding principal minors if and only if A and B are diagonally similar with transpose.*

Thus, under generic conditions, transposition and diagonal similarity are the *only* freedoms we have in finding a B such that A and B have the same set of principal minors.

As discussed further in Section 3.4, PM2MAT presently operates under the more stringent requirements that the input principal minors correspond to a matrix $A \in \mathscr{M}_n(\mathbb{C})$ which is ODF. This restriction arises since PM2MAT solves the inverse problem by examining 2×2 principal submatrices independently. Under this condition, Theorem 3.1 has the following two corollaries, which are directly relevant to the functionality of PM2MAT.

Corollary 3.2. *Let $A \in \mathscr{M}_n(\mathbb{C})$, $B, C \in \mathscr{M}_m(\mathbb{C})$ with $n > m \geqslant 2$. If A is* ODF *and $B \in \mathscr{S}_A$, then B and C have equal corresponding principal minors if and only if B and C are diagonally similar with transpose.*

Proof. One direction is straightforward. For the forward direction, assume $B = [b_{ij}]$ and C have the same corresponding principal minors. We proceed in three cases:

Case m = 2

If $B, C \in \mathscr{M}_2(\mathbb{C})$, $b_{12} \neq 0$, $b_{21} \neq 0$ and B and C have equal corresponding principal minors, then for some $t \in \mathbb{C}$, C has the form

$$C = \begin{bmatrix} b_{11} & b_{12}/t \\ b_{21}t & b_{22} \end{bmatrix}.$$

Then $C = DBD^{-1}$, where

$$D = \begin{bmatrix} 1 & 0 \\ 0 & t \end{bmatrix}.$$

Note that in this case, transposition is never necessary.

Case m = 3

Let $B, C \in \mathscr{M}_3(\mathbb{C})$. The 1×1 minors and 2×2 minors of B and C being the same means that if

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix},$$

then C has the form

$$C = \begin{bmatrix} b_{11} & b_{12}/s & b_{13}/t \\ b_{21}s & b_{22} & b_{23}/r \\ b_{31}t & b_{32}r & b_{33} \end{bmatrix}$$

for some $r, s, t \in \mathbb{C}$. Since $\det(B) = \det(C)$,

$$\det(C) - \det(B) = b_{12}b_{23}b_{31}\frac{t}{sr} - b_{12}b_{23}b_{31} + b_{13}b_{21}b_{32}\frac{sr}{t} - b_{13}b_{21}b_{32} = 0$$

since the other 4 terms of the determinants cancel. Let $c = b_{12}b_{23}b_{31}$, $d = b_{13}b_{21}b_{32}$ and $x = t/(sr)$. Because $B \in \mathscr{S}_A$ and A is ODF, all off-diagonal entries of B are nonzero which implies that $c \neq 0$ and $d \neq 0$. Solving:

$$\begin{aligned} &cx + d/x - c - d = 0 \Leftrightarrow \\ &cx^2 + (-c - d)x + d = 0 \Leftrightarrow \\ &(x-1)(cx-d) = 0 \Rightarrow \\ &x = 1 \text{ or } x = d/c. \end{aligned} \tag{3.1}$$

The first case corresponds to B being diagonally similar to C, and the second case corresponds to B being diagonally similar to C^{T}.

Case m $\geqslant$ 4

Because A is ODF, part (a) of Definition 2.3 implies that $B \in \mathscr{S}_A$ is irreducible, and part (b) of the same definition implies that the rank condition of Theorem 3.1 is satisfied. Thus, the result follows from Theorem 3.1. $\square$

Corollary 3.3. *Let* $A \in \mathscr{M}_n(\mathbb{C})$, $B, C \in \mathscr{M}_m(\mathbb{C})$ *with* $n > m \geqslant 2$ *with* A ODF *and* $B \in \mathscr{S}_A$. *If* B *and* C *have equal corresponding principal minors, then each principal submatrix of* C *is diagonally similar with transpose to the corresponding submatrix of* B; *every Schur complement of* C *is diagonally similar with transpose to the corresponding Schur complement of* B. *Consequently, each principal submatrix of every Schur complement of* C *is diagonally similar with transpose to the corresponding submatrix of the Schur complement of* B.

Proof. Without loss of generality we may confine our discussion to the principal submatrix indexed by $\alpha = \langle k \rangle$, where $0 \leqslant k \leqslant n-1$. Letting $D \in \mathscr{M}_n(\mathbb{C})$ denote a nonsingular diagonal matrix, the corollary can be verified by considering the partition

$$B = D^{-1}CD = \left[\begin{array}{c|c} D^{-1}[\alpha] & 0 \\ \hline 0 & D^{-1}(\alpha) \end{array}\right] \left[\begin{array}{c|c} C[\alpha] & C[\alpha,\alpha) \\ \hline C(\alpha,\alpha] & C(\alpha) \end{array}\right] \left[\begin{array}{c|c} D[\alpha] & 0 \\ \hline 0 & D(\alpha) \end{array}\right]$$

and the ensuing fact that

$$B/B[\alpha] = D^{-1}(\alpha)(C/C[\alpha])D(\alpha). \qquad \square$$

The practical consequence of Corollary 3.3 is that if a set of principal minors comes from a matrix A that is ODF, any matrix C that has the same corresponding principal minors as $B \in \mathscr{S}_A$ must be diagonally similar with transpose to B. Thus, each 2×2 or larger matrix encountered in the process of running MAT2PM will necessarily be diagonally similar with transpose to the corresponding submatrix, Schur complement or submatrix of a Schur complement of any matrix C that has equal corresponding principal minors to B. Therefore, the problem of finding a matrix with a given set of principal minors can be solved by finding many smaller matrices up to diagonal similarity with transpose, proceeding from level $n-1$ up to level 0 in the notation of the MAT2PM algorithm.

3.2. Description of PM2MAT

First of all, the input array *pm* of PM2MAT, consisting of the principal minors of a potential matrix $A \in \mathscr{M}_n(\mathbb{C})$, will have its entries arranged according to the following binary order. This is, in fact, the same as the order of the output of MAT2PM in [2].

Definition 3.4. Let $pm \in \mathbb{C}^{2^n-1}$ be a vector of the principal minors of $A \in \mathscr{M}_n(\mathbb{C})$. Further let i be an index of pm regarded as an n-bit binary number with

$$i = b_n b_{n-1} \dots b_3 b_2 b_1, \quad b_j \in \{0, 1\}, \quad j = 1, 2, \dots, n.$$

We say that the entries of pm are in *binary order* if

$$pm_i = \det(A[j_1, j_2, \dots, j_m]),$$

where $j_k \in \langle n \rangle$ are precisely those integers for which $b_{j_k} = 1$ for all $k = 1, 2, \dots, m$.

As a partial illustration of the binary order, we have

$$pm = [\det(A[1]), \det(A[2]), \det(A[1, 2]), \det(A[3]), \det(A[1, 3]), \det(A[2, 3]), \\ \det(A[1, 2, 3]), \det(A[4]), \dots, \det(A)].$$

3.2.1. Initial processing loop

Given input of $2^n - 1$ principal minors pm in binary order, we first produce the level $n-1$ row of 1×1 matrices. The first matrix is $[pm_{2^{n-1}}]$ which is the (n, n) entry of the desired matrix. The other matrices are found by performing the division $pm_{2^{n-1}+i}/pm_i$, $i = 1, 2, \dots, 2^{n-1} - 1$. As shown in [2], these 1×1 matrices are identical to the ones produced on the final iteration of the main level loop of MAT2PM of a matrix that had the values pm for its principal minors. For simplicity of description, we assume that division by zero never occurs, but zero principal minors can be handled by the algorithm as is described in Section 3.2.4 below. This division converts the principal minors into the 1×1 submatrices and the corresponding 1×1 Schur complements of a matrix that has pm as its principal minors.

3.2.2. Main level loop

With this initial input queue of 2^{n-1} matrices of dimension 1×1 we then enter the main level loop of PM2MAT. In general, given nq matrices of size $n1 \times n1$ (in the input queue, q), this loop produces an output queue (called qq) of $nq/2$ matrices of size $(n1 + 1) \times (n1 + 1)$. This is done in such a way that the matrices in q are either the submatrices of the matrices in qq formed by deleting their first row and column or they are the Schur complements of the matrices in qq with respect to their $(1, 1)$ entries.

By dividing the appropriate pair of principal minors we first compute the pivot or (1,1) entry of a given $(n1 + 1) \times (n1 + 1)$ matrix A we would like to create. Then, given three pieces of information: the pivot $pivot = A[1]$, the submatrix $L = A(1)$ and the Schur complement $R = A/A[1]$, we call *invschurc* to compute A. L and R are so named since submatrices are always to the left or have a smaller index in the queue q of their corresponding Schur complement in each level of the algorithm.

3.2.3. Invschurc (the main process in producing the output matrix)

Once again, recall from the description of MAT2PM in [2] (see Fig. 1) that in inverting the process in MAT2PM, one needs to reconstruct a matrix A at a given level from its Schur complement $R = A/A[1]$, as well as from $L = A(1)$ and $A[1]$ (pivot). This is achieved by *invschurc* as follows.

Let $A \in \mathscr{M}_{m+1}(\mathbb{C})$ in level $n - m - 1$, so $L, R \in \mathscr{M}_m(\mathbb{C})$ in level $n - m$. First, notice that

$$A/A[1] = A(1) - A(1, 1]\, A[1, 1)/A[1] \;\Leftrightarrow$$

$$L - R = A(1) - A/A[1] = A(1, 1]\, A[1, 1)/A[1]. \tag{3.2}$$

If $\operatorname{rank}(L-R)=1$, we can find vectors $A(1,1]$ and $A[1,1)$ such that $A(1,1]A[1,1)=(L-R)A[1]$ by setting

$$A[1,1)=(L-R)[i,\langle m\rangle], \tag{3.3}$$

which, in turn, implies that

$$A(1,1]=(L-R)[\langle m\rangle,i]A[1]/(L-R)[i,i], \tag{3.4}$$

where i is chosen such that

$$|(L-R)_{ii}|=\max_{j\in\langle m\rangle}|(L-R)_{jj}| \tag{3.5}$$

to avoid division by a small quantity. Similar to partial pivoting in Gaussian elimination, this will reduce cancellation errors when the output matrix A is used in a difference with another matrix computed using the same convention.

The difficult part of this computation is that, in general, the difference of L and R as input to *invschurc* has rank higher than 1. For input matrices L and R that are larger than 2×2, this problem is solved by invoking part (c) of Definition 2.3 to enable a diagonal similarity with transpose of R to be found that makes $\operatorname{rank}(L-R)=1$ by finding the unique dot similarity with R that satisfies the rank condition.

We consider four cases:

Case m = 1

In this case $\operatorname{rank}(L-R)=1$ trivially, so the computation above suffices to produce A.

Case m = 2

If *invschurc* is called with $L, R\in\mathscr{M}_2(\mathbb{C})$, $L-R$ will usually not be rank 1. This is because when $L=[l_{ij}]$ and $R=[r_{ij}]$ were produced by prior calls to *invschurc*, the exact values of l_{12}, l_{21}, r_{12} and r_{21} were not known. Only the products $l_{12}l_{21}$ and $r_{12}r_{21}$ were fixed by knowing the given Schur complement. We can remedy this by choosing $t\in\mathbb{C}$, $t\neq 0$ such that

$$\operatorname{rank}\left(\begin{bmatrix} l_{11} & l_{12}\\ l_{21} & l_{22}\end{bmatrix}-\begin{bmatrix} r_{11} & r_{12}/t\\ r_{21}t & r_{22}\end{bmatrix}\right)=1. \tag{3.6}$$

Note that modifying R in this way is a diagonal similarity with the nonsingular diagonal matrix

$$D=\begin{bmatrix}1 & 0\\ 0 & t\end{bmatrix},$$

so all the principal minors of R remain unchanged. Also note that in this case, diagonal similarity and dot similarity are equivalent.

Solving

$$\frac{l_{11}-r_{11}}{l_{21}-r_{21}t}=\frac{l_{12}-r_{12}/t}{l_{22}-r_{22}}$$

for t we obtain two solutions to the quadratic equation

$$t_1,t_2=(-x_1x_2+l_{12}l_{21}+r_{12}r_{21}\pm\sqrt{d})/(2l_{12}r_{21}), \tag{3.7}$$

where

$$x_1=l_{11}-r_{11},\quad x_2=l_{22}-r_{22}$$

and

$$d = x_1^2 x_2^2 + l_{12}^2 l_{21}^2 + r_{12}^2 r_{21}^2 - 2x_1 x_2 l_{12} l_{21} - 2x_1 x_2 r_{12} r_{21} - 2 l_{12} l_{21} r_{21} r_{12}.$$

This algebra has been placed in the subroutine *solveright* of PM2MAT in Appendix A.

The choice of t_1 versus t_2 is arbitrary, so we always choose t_1. Choosing t_2 instead for all $m = 2$ matrices merely results in the final output matrix of PM2MAT being the transpose of a diagonal similarity to what it would have been otherwise.

Remark 3.5. If $L, R \in \mathscr{M}_2(\mathbb{R})$ and R is diagonally similar to a matrix $\hat{R} \in \mathscr{M}_2(\mathbb{R})$ such that $\operatorname{rank}(L - \hat{R}) = 1$, then d above will be non-negative by virtue of the construction of the quadratic system (3.7). Although d factors somewhat as

$$d = l_{12}^2 l_{21}^2 - 2 l_{12} l_{21}(r_{12} r_{21} + x_1 x_2) + (r_{12} r_{21} - x_1 x_2)^2,$$

d is not a sum of perfect squares and d can be negative with real parameters if the rank condition does not hold. Also note that, in general, if the principal minors come from a matrix which is ODF, division by zero in (3.7) never occurs.

Remark 3.6. Parenthetically to the ongoing analysis, note that in general if L and R have the same principal minors as $\hat{L}$ and $\hat{R}$ respectively, then, under the conditions that PM2MAT operates, Corollary 3.3 implies that ($\hat{L} = D_L L D_L^{-1}$ or $\hat{L} = D_L L^{\mathrm{T}} D_L^{-1}$) and ($\hat{R} = D_R R D_R^{-1}$ or $\hat{R} = D_R R^{\mathrm{T}} D_R^{-1}$) for diagonal matrices D_L and D_R. Therefore, either $\operatorname{rank}(D_L L D_L^{-1} - D_R R D_R^{-1}) = 1$ or $\operatorname{rank}(D_L L D_L^{-1} - D_R R^{\mathrm{T}} D_R^{-1}) = 1$. Without loss of generality, if

$$\operatorname{rank}(D_L L D_L^{-1} - D_R R D_R^{-1}) = 1$$

then

$$\operatorname{rank}(L - D_L^{-1} D_R R D_R^{-1} D_L) = \operatorname{rank}(L - D R D^{-1}) = 1,$$

where $D = D_L^{-1} D_R$. Thus, finding a diagonal similarity with transpose of the right matrix suffices to make $L - R$ rank 1 even if *both* L and R are diagonally similar with transpose to matrices whose difference is rank 1.

Case $\mathbf{m = 3}$

If *invschurc* is called with $L, R \in \mathscr{M}_3(\mathbb{C})$, we attempt to find $r, s, t \in \mathbb{C}$ such that

$$\operatorname{rank}\left(\begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix} - \begin{bmatrix} r_{11} & r_{12}/s & r_{13}/t \\ r_{21}s & r_{22} & r_{23}/r \\ r_{31}t & r_{32}r & r_{33} \end{bmatrix}\right) = 1. \tag{3.8}$$

This is done by finding the two quadratic solutions for each 2×2 submatrix of $L - R$ for r, s and t independently. Since $r = \{r_1, r_2\}$, $s = \{s_1, s_2\}$ and $t = \{t_1, t_2\}$, there are 8 possible combinations of parameters to examine which correspond to 8 possible dot similarities with R. Part (c) of Definition 2.3 guarantees that only one of the combinations will make $\operatorname{rank}(L - R) = 1$. Since

$$\frac{l_{11} - r_{11}}{l_{21} - r_{21}s} = \frac{l_{13} - r_{13}/t}{l_{23} - r_{23}/r} \Rightarrow (l_{23} - r_{23}/r)(l_{11} - r_{11}) - (l_{21} - r_{21}s)(l_{13} - r_{13}/t) = 0,$$

we compare

$$|(l_{23} - r_{23}/r)(l_{11} - r_{11}) - (l_{21} - r_{21}s)(l_{13} - r_{13}/t)| \tag{3.9}$$

for each combination of r, s, t. Note that if the principal minors come from a matrix which is ODF, this dot similarity will compute the unique diagonal similarity with transpose of R necessary to make $\operatorname{rank}(L - R) = 1$.

If multiple combinations of solutions result in expressions of the form of (3.9) that are (nearly) zero, then a warning is printed to indicate that the output of PM2MAT is suspect. An example of a matrix A that satisfies Parts (a) and (b) of Definition 2.3 but fails to satisfy Part (c) is presented in **Case (c)** of Section 3.4.

Case m > 3

If *invschurc* is called with $L, R \in \mathscr{M}_m(\mathbb{C})$, $m > 3$, we make $L - R$ of rank 1 by modifying R starting at the lower right hand corner and working to the upper left. This is better explained by appealing to the $m = 5$ case: Consider $L - R \in \mathscr{M}_5(\mathbb{C})$ in the form

$$L - \begin{bmatrix} r_{11} & r_{12}/s^{(4)} & r_{13}/t^{(4)} & r_{14}/t^{(5)} & r_{15}/t^{(6)} \\ r_{21}s^{(4)} & r_{22} & r_{23}/s^{(2)} & r_{24}/t^{(2)} & r_{25}/t^{(3)} \\ r_{31}t^{(4)} & r_{32}s^{(2)} & r_{33} & r_{34}/s^{(1)} & r_{35}/t^{(1)} \\ r_{41}t^{(5)} & r_{42}t^{(2)} & r_{43}s^{(1)} & r_{44} & r_{45}/r^{(1)} \\ r_{51}t^{(6)} & r_{52}t^{(3)} & r_{53}t^{(1)} & r_{54}r^{(1)} & r_{55} \end{bmatrix}. \tag{3.10}$$

We first find values for $r^{(1)}$, $s^{(1)}$ and $t^{(1)}$ for the lower right 3×3 submatrix of $L - R$ as in the $m = 3$ section above. Then, by examining four combinations of parameters with two potentially different solutions for s and t, we find $s^{(2)}$ and $t^{(2)}$ by seeing which combination of solutions causes the lower right 4×4 submatrix to be of rank 1. Again, Part (c) of Definition 2.3 is invoked to know that only one combination of solutions will satisfy the rank condition. We choose $t^{(3)}$ by again requiring that the lower right 4×4 submatrix be rank 1, choosing one out of two possibly different solutions.

It is tempting to use

$$\frac{l_{22} - r_{22}}{l_{32} - r_{32}s^{(2)}} = \frac{l_{25} - r_{25}/t^{(3)}}{l_{35} - r_{35}/t^{(1)}}$$

to find $t^{(3)}$. Accumulating inaccuracies, however, prevent this from working well when inverting vectors of principal minors into larger (than 10×10) matrices. The remaining parameters are found in the order their superscripts imply.

3.2.4. Handling zero principal minors

The basic algorithm of MAT2PM applied to a given $A \in \mathscr{M}_n(\mathbb{C})$ can only proceed as long as the (1,1) or pivot entry of each matrix in $\mathscr{S}_A$ is nonzero. However, it was found (see [2, Example 5.1, Zero Pivot Loop]) that the algorithm of MAT2PM can be extended to proceed in this case by using a different pivot value (called a pseudo-pivot or *ppivot* in the code). Then, the multilinearity of the determinant implies that the desired principal minors are differences of the principal minors which are computed using *ppivot* for all zero pivot values. These corrections to make the principal minors correspond to the principal minors of the input matrix A are done in the zero pivot loop at the end of MAT2PM.

At the beginning of PM2MAT there is a loop analogous to the zero pivot loop at the end of MAT2PM. A principal minor in the vector *pm* being zero corresponds to the (1,1) or pivot entry of an intermediate matrix computed by PM2MAT being zero. We can modify a zero principal minor so that instead, the given matrix will have a (1, 1) entry equal to *ppivot*, which makes the principal minor nonzero also. Since only the first $2^{n-1} - 1$ principal minors in the array of $2^n - 1$ principal minors appear in the denominator when computing pivots of intermediate matrices, this operation is only done for these initial principal minors. Thus, division by zero never occurs when taking ratios of principal minors in PM2MAT.

A random constant value was chosen for *ppivot* in PM2MAT to reduce the chance that an off-diagonal zero will result when submatrices and Schur complements are constructed in *invschurc* using *ppivot* as the pivot value. Thus, it is more likely that the resulting principal minors come from a matrix which is effectively ODF. By *effectively ODF* we mean a matrix $A \in \mathscr{M}_n(\mathbb{C})$ that satisfies all three conditions of Definition 2.3 with the set $\mathscr{S}_A$ computed as follows: If a given Schur complement exists, it is computed as usual. Otherwise, the Schur complement is computed with the (1, 1) entry of the matrix set equal to *ppivot*. The submatrices in $\mathscr{S}_A$ are also found as usual.

Applying the multilinearity of the determinant, all later principal minors in the *pm* vector that are affected by changing a given zero principal minor to a nonzero value are modified, and the index of the principal minor that was changed is stored in the vector of indices *zeropivs*. Later, when we use this principal minor as the numerator of a *pivot*, we subtract *ppivot* from the (1, 1) entry of the resulting matrix to produce a final matrix having the desired zero principal minors.

For the simple case that the zero principal minors of A correspond to zero diagonal entries of A, this process amounts to adding a constant to the given zero principal minor, then subtracting the same constant from the corresponding diagonal entry of A.

An illustration to show how a zero diagonal entry of a Schur complement can be correctly handled is given in Example 4.2.

3.2.5. Deskewing

The numerical output of PM2MAT is a matrix which has (nearly) the same principal minors as the input vector of principal minors *pm*. However, since principal minors are not changed by diagonal similarity, the output of PM2MAT may not be very presentable in terms of the dynamic range of its off-diagonal entries. In particular, if a random matrix was used to generate the principal minors, the output of PM2MAT with those principal minors will generally have a significantly higher condition number than the original input matrix. To address this issue, at the end of PM2MAT we perform a diagonal similarity DAD^{-1}, referred to as *deskewing*. Let D be an $n \times n$ diagonal matrix with positive diagonal entries where for convenience the (1, 1) entry of D is 1. Then, d_{ii} is chosen such that $|a_{i1}d_{ii}| = |a_{1i}/d_{ii}|$ for all $i = 2, 3, \ldots, n$; that is

$$d_{ii} = \sqrt{\left|\frac{a_{1i}}{a_{i1}}\right|}, \quad i = 2, 3, \ldots, n.$$

If a_{i1} is zero or the magnitude of d_{ii} is deemed to be too large or too close to zero, $d_{ii} = 1$ is used instead.

3.2.6. PM2MAT in the field of real numbers

The natural field of operation of MAT2PM and PM2MAT is $\mathbb{C}$, the field of complex numbers. If PM2MAT is run on principal minors that come from a real ODF matrix, then the resulting matrix will also be real (see Remark 3.5). It is possible, however, that a real set of principal minors does not correspond to a real matrix and so PM2MAT produces a non-real matrix.

More specifically, although for every set of three real numbers there exists a real matrix in $\mathscr{M}_2(\mathbb{R})$ that has them as principal minors, this result does not generalize for $n \times n$ matrices when $n > 2$. For instance, if

$$A = \begin{bmatrix} 1 & \mathrm{i} & 5 \\ -\mathrm{i} & 2 & 1 \\ 5 & 1 & 3 \end{bmatrix},$$

then A is ODF and the principal minors of A in binary order are

$$pm = [1, 2, 1, 3, -22, 5, -48],$$

which are certainly real. From the argument of **Case n = 3** of Corollary 3.2, any matrix which has the same principal minors as A must be diagonally similar (with transpose) to A. Transposition does not affect the field that A resides in. Without loss of generality consider a diagonal similarity of A by a diagonal matrix D with $d_{11} = 1$. Then

$$DAD^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix} \begin{bmatrix} 1 & \mathrm{i} & 5 \\ -\mathrm{i} & 2 & 1 \\ 5 & 1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{d_{22}} & 0 \\ 0 & 0 & \frac{1}{d_{33}} \end{bmatrix} = \begin{bmatrix} 1 & \frac{\mathrm{i}}{d_{22}} & \frac{5}{d_{33}} \\ -\mathrm{i}d_{22} & 2 & \frac{d_{22}}{d_{33}} \\ 5d_{33} & \frac{d_{33}}{d_{22}} & 3 \end{bmatrix}$$

is non-real for all $d_{22}, d_{33} \in \mathbb{C}$. That is, $pm \in \mathbb{R}^7$ above can only correspond to a non-real matrix and such will be the output of PM2MAT.

3.3. Operation count

The approximate number of floating point operations required by PM2MAT can be found by counting the operations required to perform the following two primary inner loop tasks:

(a) solving the quadratics (3.7) in *solveright* and
(b) evaluating equations of the form (3.9) to select the appropriate quadratic solution.

Task (a)
Solving (3.7) is done once for every entry in the upper triangular part of every matrix in level $n-2$ through level 0. So, for an $m \times m$ matrix, there will be $m(m-1)/2$ quadratics solved. Letting q be the number of floating point operations to solve the quadratic (which is approximately 40), the total number of floating point operations in PM2MAT solving quadratics is

$$\sum_{k=0}^{n-3} 2^k (n-(k+1))(n-(k+2))q/2 = q \cdot 2^n - (q/2)(n^2+n+2), \tag{3.11}$$

which is of $O(2^n)$.
Task (b)
Let p be the number of floating point operations to evaluate Eq. (3.9) (which is approximately 11). For a given $m \times m$ matrix, $m \geqslant 3$, there are eight evaluations of (3.9) for the lower right 3×3 submatrix (to choose the correct $r^{(1)}, s^{(1)}, t^{(1)}$ of (3.10)), $4(m-3)$ evaluations (to find $s^{(2)}, t^{(2)}, s^{(4)}, t^{(4)}$ of (3.10)) and $2(m-2)(m-3)/2 = (m-2)(m-3)$ evaluations to find the remaining parameters below the second subdiagonal of each matrix (parameters $t^{(3)}, t^{(5)}, t^{(6)}$ of (3.10)). Thus, the total number of operations needed for these computations is

$$p \sum_{k=0}^{n-4} 2^k (8 + 4(n-(k+4)) + (n-(k+3))(n-(k+4))) = p(2 \cdot 2^n - (n^2+n+4)). \tag{3.12}$$

Since this is also of $O(2^n)$, the total operation count is $O(2^n)$. Other incidental computations (various mins, building the $(m+1) \times (m+1)$ matrix at the end of *invschurc*, etc.) have no effect on the order of the computation and do not significantly add to the total number of operations.

3.4. Strategy for solving PMAP via PM2MAT and MAT2PM

First, we illustrate that if $A \in \mathscr{M}_n(\mathbb{C})$ is not ODF, then the principal minors of A may not be invertible by PM2MAT into a matrix. We consider examples that violate each part of Definition 2.3.

Case (a): An off-diagonal entry of a matrix in $\mathscr{S}_{\mathbf{A}}$ is zero
Suppose we use PM2MAT to find a matrix with the principal minors of

$$A = \begin{bmatrix} -4 & -3 & -8 \\ 2 & 3 & 5 \\ 8 & 6 & 7 \end{bmatrix}.$$

Since the (2, 1) entry of $A/A[1]$ is zero, we could run into difficulties. Running the MAT2PM algorithm on this matrix we obtain:

$$A = \begin{bmatrix} -4 & -3 & -8 \\ 2 & 3 & 5 \\ 8 & 6 & 7 \end{bmatrix}, \qquad \text{(Level 0)}$$

$$L_A = \begin{bmatrix} 3 & 5 \\ 6 & 7 \end{bmatrix}, \quad R_A = \begin{bmatrix} \frac{3}{2} & 1 \\ 0 & -9 \end{bmatrix}, \qquad \text{(Level 1)}$$

Running PM2MAT on the principal minors of A, we see that PM2MAT is able to produce matrices diagonally similar to L_A and R_A at its level 1.

$$L_B = \begin{bmatrix} 3 & 10 \\ 3 & 7 \end{bmatrix}, \quad R_B = \begin{bmatrix} \frac{3}{2} & 1 \\ 0 & -9 \end{bmatrix}, \qquad \text{(Level 1)}$$

Unfortunately, the zero in the (2, 1) entry of R_B causes a divide by zero in Eq. (3.7). Therefore, PM2MAT is unable to find the diagonal similarity that makes $\operatorname{rank}(L_B - R_B) = 1$, and PM2MAT will not produce an output matrix with the desired principal minors. Although it is possible to add heuristics to PM2MAT to deal with this particular example, in general, the algebraic methods employed in *invschurc* (see particularly (3.7) and (3.9) in Section 3.2.3) break down when off-diagonal zeros occur in any of the matrices generated by PM2MAT.

Case (b): Rank condition for partitions of $\langle m \rangle$ not satisfied for matrices in $\mathscr{S}_{\mathbf{A}}$
Consider the first two levels of the MAT2PM algorithm with the following input matrix A:

$$A = \begin{bmatrix} 2 & -3 & 7 & 1 & 1 \\ 4 & 5 & -3 & 1 & 1 \\ 4 & -4 & 13 & 1 & 1 \\ 1 & 1 & 1 & 8 & 2 \\ 1 & 1 & 1 & 7 & 5 \end{bmatrix}, \qquad \text{(Level 0)}$$

$$L_A = \begin{bmatrix} 5 & -3 & 1 & 1 \\ -4 & 13 & 1 & 1 \\ 1 & 1 & 8 & 2 \\ 1 & 1 & 7 & 5 \end{bmatrix}, \quad R_A = \begin{bmatrix} 11 & -17 & -1 & -1 \\ 2 & -1 & -1 & -1 \\ \frac{5}{2} & -\frac{5}{2} & \frac{15}{2} & \frac{3}{2} \\ \frac{5}{2} & -\frac{5}{2} & \frac{13}{2} & \frac{9}{2} \end{bmatrix}. \qquad \text{(Level 1)}$$

Note that

$$\operatorname{rank}(L_A[\{1, 2\}, \{3, 4\}]) = \operatorname{rank}(L_A[\{3, 4\}, \{1, 2\}]) = 1,$$
$$\operatorname{rank}(R_A[\{1, 2\}, \{3, 4\}]) = \operatorname{rank}(R_A[\{3, 4\}, \{1, 2\}]) = 1,$$

where $\alpha = \{1, 2\}$ and $\beta = \{3, 4\}$ form a partition of $\langle 4 \rangle$.

When PM2MAT is run with the principal minors of A as input, we produce the following matrices for level 1:

$$L_B = \begin{bmatrix} 5 & \frac{12}{5} & -\frac{4}{65} & -\frac{1}{65} \\ 5 & 13 & \frac{1}{13} & \frac{1}{52} \\ -\frac{65}{4} & 13 & 8 & \frac{7}{4} \\ -65 & 52 & 8 & 5 \end{bmatrix}, \quad R_B = \begin{bmatrix} 11 & -\frac{34}{11} & -\frac{5}{11} & -\frac{13}{33} \\ 11 & -1 & -\frac{5}{2} & -\frac{13}{6} \\ \frac{11}{2} & -1 & \frac{15}{2} & \frac{13}{10} \\ \frac{165}{26} & -\frac{15}{13} & \frac{15}{2} & \frac{9}{2} \end{bmatrix}. \qquad \text{(Level 1)}$$

One may verify that L_A and L_B have the same principal minors. Likewise, R_A and R_B have the same principal minors. However, since the conditions of Theorem 3.1 are not satisfied for L_A, L_B need not be diagonally similar with transpose to L_A, and indeed this is the case. When solving the 3 sets of quadratics to make the lower right 3×3 submatrix of $L_B - R_B$ rank 1, there are two combinations of solutions that both make $\operatorname{rank}(L_B(1) - R_B(1)) = 1$. By chance, the wrong solution is chosen, so the computation fails for the matrices as a whole. Solving the quadratics of Eq. (3.7) for the rest of the entries of the matrices does not yield a combination of solutions that makes $\operatorname{rank}(L_B - R_B) = 1$.

By coincidence, R_A and R_B are diagonally similar with transpose, but the L matrices not being diagonally similar with transpose is sufficient to cause the computations of *invschurc* to fail.

Although this example suggests that (c) may imply (b) under the condition of (a) of Definition 2.3, the converse is certainly not true as the following example shows.

Case (c): There exist multiple solutions which make rank(L − R)= 1

Similar to the last case, the operation of MAT2PM on $A \in \mathscr{M}_4(\mathbb{R})$ is summarized below down to level 2.

$$A = \begin{bmatrix} 3 & -6 & -9 & 12 \\ 1 & 3 & -7 & -\frac{4}{3} \\ 2 & 2 & -1 & -\frac{16}{9} \\ 3 & 1 & 1 & 4 \end{bmatrix}, \qquad \text{(Level 0)}$$

$$L_A = \begin{bmatrix} 3 & -7 & -\frac{4}{3} \\ 2 & -1 & -\frac{16}{9} \\ 1 & 1 & 4 \end{bmatrix}, \quad R_A = \begin{bmatrix} 5 & -4 & -\frac{16}{3} \\ 6 & 5 & -\frac{88}{9} \\ 7 & 10 & -8 \end{bmatrix}, \qquad \text{(Level 1)}$$

$$\begin{bmatrix} -1 & -\frac{16}{9} \\ 1 & 4 \end{bmatrix}, \quad \begin{bmatrix} 5 & -\frac{88}{9} \\ 10 & -8 \end{bmatrix}, \quad \begin{bmatrix} \frac{11}{3} & -\frac{8}{9} \\ \frac{10}{3} & \frac{40}{9} \end{bmatrix}, \quad \begin{bmatrix} \frac{49}{5} & -\frac{152}{45} \\ \frac{78}{5} & -\frac{8}{15} \end{bmatrix}. \qquad \text{(Level 2)}$$

Running PM2MAT on the principal minors of A above yields:

$$\begin{bmatrix} -1 & \frac{16}{9} \\ -1 & 4 \end{bmatrix}, \quad \begin{bmatrix} 5 & -\frac{176}{9} \\ 5 & -8 \end{bmatrix}, \quad \begin{bmatrix} \frac{11}{3} & -\frac{80}{99} \\ \frac{11}{3} & \frac{40}{9} \end{bmatrix}, \quad \begin{bmatrix} \frac{49}{5} & -\frac{3952}{735} \\ \frac{49}{5} & -\frac{8}{15} \end{bmatrix}, \qquad \text{(Level 2)}$$

$$L_B = \begin{bmatrix} 3 & -\frac{14}{3} & \frac{8}{9} \\ 3 & -1 & \frac{16}{9} \\ -\frac{3}{2} & -1 & 4 \end{bmatrix}, \quad R_B = \begin{bmatrix} 5 & -\frac{14}{5} & -\frac{112}{15} \\ \frac{60}{7} & 5 & -\frac{176}{9} \\ 5 & 5 & -8 \end{bmatrix}, \qquad \text{(Level 1)}$$

$$B = \begin{bmatrix} 3 & \frac{9}{2} & \frac{9}{2} & 12 \\ -\frac{4}{3} & 3 & -\frac{14}{3} & \frac{8}{9} \\ -4 & 3 & -1 & \frac{16}{9} \\ 3 & -\frac{3}{2} & -1 & 4 \end{bmatrix}. \tag{Level 0}$$

Although L_A is diagonally similar to L_B and R_A is diagonally similar to R_B, there are two solutions that make $\operatorname{rank}(L_B - R_B) = 1$. Instead of choosing the dot similarity with R_B

$$R_1 = \begin{bmatrix} 5 & -\frac{8}{3} & \frac{32}{9} \\ 9 & 5 & \frac{88}{9} \\ -\frac{21}{2} & -10 & -8 \end{bmatrix}$$

which has determinant 712/3 like R_A and R_B, the dot similarity

$$R_2 = \begin{bmatrix} 5 & -\frac{8}{3} & \frac{56}{9} \\ 9 & 5 & \frac{160}{9} \\ -6 & -\frac{11}{2} & -8 \end{bmatrix}$$

with determinant 260 is chosen, which also satisfies $\operatorname{rank}(L_B - R_2) = 1$. Therefore, the resulting B which is computed with R_2 above is not diagonally similar to A. In fact, B is diagonally similar to

$$C = \begin{bmatrix} 3 & \boxed{-3} & \boxed{-9/2} & 12 \\ \boxed{2} & 3 & -7 & -4/3 \\ \boxed{4} & 2 & -1 & -16/9 \\ 3 & 1 & 1 & 4 \end{bmatrix},$$

where only the boxed entries differ from A. The principal minors of C (and B) are the same as the principal minors of A with the exception of the determinant. Note that the output of PM2MAT in this example is dependent on small round-off errors which may vary from platform to platform.

Although it would be easy to add code to PM2MAT to resolve this ambiguity for the case that $L, R \in \mathscr{M}_3(\mathbb{C})$, in general, the method of *invschurc* which finds solutions for making $\operatorname{rank}(L - R) = 1$ by working from the trailing 3×3 submatrix of L and R and progressing towards the upper right (see Eq. (3.10)) is not well suited to also preserving diagonal similarity with transpose in cases where (c) of Definition 2.3 is not satisfied.

If the input principal minors do come from a matrix which is ODF, then PM2MAT will succeed in building a matrix that has them as principal minors up to numerical limitations. The following proposition states this formally:

Proposition 3.7. *Let* $pm \in \mathbb{C}^{2^n-1}$, $n \in \mathbb{N}$ *be given. Suppose* $A \in \mathscr{M}_n(\mathbb{C})$ *is* ODF. *If* $\mathrm{MAT2PM}(A) = pm$ *and* $B = \mathrm{PM2MAT}(pm)$, *then* $\mathrm{MAT2PM}(B) = pm$.

Proof. Let $C \in \mathscr{M}_m(\mathbb{C}), m \geqslant 2$ be any submatrix or Schur complement produced by running MAT2PM on A ($C \in \mathscr{S}_A$). Since A is ODF, C has no off-diagonal zero entries and thus

$$L - R = C(1) - C/C[1] = C(1, 1]C[1, 1)/C[1]$$

has no zero entries. Hence, $\operatorname{rank}(L - R) \neq 0$ for all such C.

Level $n-1$ can always be created which matches the output of level $n-1$ of MAT2PM(A) exactly, since this level of 1×1 matrices just consists of ratios of principal minors, where zero principal minors can be handled using the multilinearity of the determinant (see Section 3.2.4).

For all the 2×2 matrices of level $n-2$, $\operatorname{rank}(L-R) \neq 0$ for each matrix we produce, so we are able to compute matrices which have the same principal minors as each of the corresponding 2×2 matrices of level $n-2$ we get from running MAT2PM on A. By Corollary 3.3, the matrices produced by PM2MAT are diagonally similar with transpose to the ones produced by MAT2PM, so by solving equations of the form of (3.7) (which always have solutions since if A is ODF, division by zero never occurs) we can modify R so $\operatorname{rank}(L-R) = 1$ for each matrix of this level.

For subsequent levels, conditions (a) and (b) of Definition 2.3 guarantee that a diagonal similarity with transpose for all the R matrices exists so that $\operatorname{rank}(L-R) = 1$ for each L, R pair. Furthermore, condition (c) of the same definition guarantees that the dot similarity found by *invschurc* is sufficient to find this (unique, if L is regarded as fixed) similarity.

Proceeding inductively we see that the final matrix B output by PM2MAT has the same principal minors as A, since B has the same (1,1) entry as A, $L = B(1)$ is diagonally similar with transpose to $A(1)$ and $R = B/B[1]$ is diagonally similar with transpose to $A/A[1]$. The MAT2PM algorithm can then be used to verify that B and A have identical corresponding principal minors. $\square$

Note that identical reasoning extends the result above to effectively ODF matrices.

If the input principal minors do not come from an effectively ODF matrix, PM2MAT may fail to produce a matrix with the desired principal minors. If the input principal minors are inconsistent, that is, a matrix which has them as principal minors does not exist, PM2MAT will always fail to produce a matrix with the input principal minors.

When PM2MAT fails due to either inconsistent principal minors or due to principal minors from an non-ODF matrix, it produces a matrix which does not have all the desired principal minors; no warning that this has occurred is guaranteed, since the warnings in PM2MAT depend upon numerical thresholds.

This suggests the following strategy for attempting to solve PMAP with principal minors from an unknown source with PM2MAT.

1. PM2MAT is run on the input principal minors and an output matrix A is produced.
2. Then, MAT2PM is run on the matrix A, producing a second set of principal minors.
3. These principal minors are compared to the original input principal minors. If they agree to an acceptable tolerance, PM2MAT succeeded. Otherwise, the principal minors are either inconsistent or they are not realizable by a matrix which is effectively ODF.

A sample implementation of this strategy is provided in PMFRONT in Appendix B.

4. Illustrative examples

Example 4.1. We obtain a moderately scaled set of integer principal minors that correspond to a real matrix. To do this, we use MAT2PM to find the principal minors of

$$A = \begin{bmatrix} -6 & 3 & -9 & 4 \\ -6 & -5 & 3 & 6 \\ 3 & -3 & 6 & -7 \\ 1 & 1 & -1 & -3 \end{bmatrix},$$

yielding

$$pm = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ [1] & [2] & [1,2] & [3] & [1,3] & [2,3] & [1,2,3] \\ -6 & -5 & 48 & 6 & -9 & -21 & -36 \end{pmatrix.$$
$$\begin{matrix} 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ [4] & [1,4] & [2,4] & [1,2,4] & [3,4] & [1,3,4] & [2,3,4] & [1,2,3,4] \\ -3 & 14 & 9 & -94 & -25 & 96 & 59 & 6 \end{matrix}\right).$$

For ease of reference, the first row of the above displayed pm contains the index of a given principal minor, the second row is the index set used to obtain the submatrix corresponding to that minor, while the third row contains the value of the principal minor itself. For convenience, the utility PMSHOW in Appendix G has been included, which produces output similar to this for any input set of principal minors. To actually call PM2MAT, pm is just the vector of values in the third row. Thus we can easily see that $pm_{13} = \det(A[1,3,4]) = 96$, for example.

Since there are $2^4 - 1$ entries in pm, we desire to find a matrix $B \in \mathscr{M}_4(\mathbb{R})$ which has the values in pm as its principal minors. We begin with $level = 3$ and produce matrices equivalent in all principal minors at each level to the ones that would have been produced by running MAT2PM on A.

Level 3

At level 3 we desire to find eight 1×1 matrices whose entries are just the pivots in binary order. Since the principal minors are stored in binary order (see Definition 3.4 and [2, Proposition 4.6]) we compute the following quotients of consecutive principal minors:

$$pm_8 = \det(A[4]) = -3,$$
$$\frac{pm_9}{pm_1} = \frac{\det(A[1,4])}{\det(A[1])} = -\frac{7}{3}, \ \ldots, \ \frac{pm_{15}}{pm_7} = \frac{\det(A[1,2,3,4])}{\det(A[1,2,3])} = -\frac{1}{6}.$$

Therefore, the desired level 3 matrices are

$$[-3], \quad \left[-\tfrac{7}{3}\right], \quad \left[-\tfrac{9}{5}\right], \quad \left[-\tfrac{47}{24}\right], \quad \left[-\tfrac{25}{6}\right], \quad \left[-\tfrac{32}{3}\right], \quad \left[-\tfrac{59}{21}\right], \quad \left[-\tfrac{1}{6}\right].$$

Level 2

Analogously to the previous level, we can now compute the four pivots of the four 2×2 matrices from the eight matrices of the previous level:

$$pm_4 = 6, \ \frac{pm_5}{pm_1} = \frac{3}{2}, \ \frac{pm_6}{pm_2} = \frac{21}{5}, \ \frac{pm_7}{pm_3} = -\frac{3}{4}.$$

Since the first four 1×1 matrices of the previous level are submatrices of the four matrices we desire to compute, we know that these matrices have the form

$$B = \begin{bmatrix} 6 & * \\ * & -3 \end{bmatrix}, \quad \begin{bmatrix} \frac{3}{2} & * \\ * & -\frac{7}{3} \end{bmatrix}, \quad \begin{bmatrix} \frac{21}{5} & * \\ * & -\frac{9}{5} \end{bmatrix}, \quad \begin{bmatrix} -\frac{3}{4} & * \\ * & -\frac{47}{24} \end{bmatrix},$$

where the $*$'s indicate entries not yet known. Consider producing the first of these matrices, labeled B and partitioned as follows:

$$B = \left[\begin{array}{c|c} B[1] & B[1,1) \\ \hline B(1,1] & B(1) \end{array}\right].$$

From level 3 we know $L = B[1] = -3$ and $R = B/B[1] = -25/6$. Following Eq. (3.2),

$$L - R = -3 + 25/6 = 7/6 = B(1, 1]B[1, 1)/B[1].$$

Taking $B[1, 1) = 7/6$, we satisfy the equation above with $B(1, 1] = B[1]$. Thus we have computed

$$B = \begin{bmatrix} 6 & \frac{7}{6} \\ 6 & -3 \end{bmatrix}.$$

Repeating this procedure for the other three matrices we finish level 2 with the following four matrices:

$$\begin{bmatrix} 6 & \frac{7}{6} \\ 6 & -3 \end{bmatrix}, \quad \begin{bmatrix} \frac{3}{2} & \frac{25}{3} \\ \frac{3}{2} & -\frac{7}{3} \end{bmatrix}, \quad \begin{bmatrix} \frac{21}{5} & \frac{106}{105} \\ \frac{21}{5} & -\frac{9}{5} \end{bmatrix}, \quad \begin{bmatrix} -\frac{3}{4} & -\frac{43}{24} \\ -\frac{3}{4} & -\frac{47}{24} \end{bmatrix}.$$

Level 1

Computing the two pivots for the matrices of this level, we get $pm_2 = -5$ and $pm_3/pm_1 = 48/-6 = -8$. Thus, we seek to complete the matrices

$$C = \begin{bmatrix} -5 & * & * \\ * & 6 & \frac{7}{6} \\ * & 6 & -3 \end{bmatrix}, \quad \begin{bmatrix} -8 & * & * \\ * & \frac{3}{2} & \frac{25}{3} \\ * & \frac{3}{2} & -\frac{7}{3} \end{bmatrix}.$$

We label the first of these two matrices C, and we call *invschurc* with

$$L = C(1) = \begin{bmatrix} 6 & \frac{7}{6} \\ 6 & -3 \end{bmatrix}, \quad R = C/C[1] = \begin{bmatrix} \frac{21}{5} & \frac{106}{105} \\ \frac{21}{5} & -\frac{9}{5} \end{bmatrix}$$

from the previous level. Since

$$L - R \approx \begin{bmatrix} 1.8 & .1571 \\ 1.8 & -1.2 \end{bmatrix}$$

is not rank one, we seek a $t \in \mathbb{R}$ such that Eq. (3.6) is satisfied. Solving Eq. (3.7), we find solutions $t_1 = 4/7$, $t_2 = 106/49$. Using t_1,

$$L - R = \begin{bmatrix} 6 & \frac{7}{6} \\ 6 & -3 \end{bmatrix} - \begin{bmatrix} \frac{21}{5} & \frac{53}{30} \\ \frac{12}{5} & -\frac{9}{5} \end{bmatrix} = \begin{bmatrix} \frac{9}{5} & -\frac{3}{5} \\ \frac{18}{5} & -\frac{6}{5} \end{bmatrix}.$$

Letting $C[1, 1) = (L - R)[\{1\}, \{1, 2\}] = [-9/5, -3/5]$, we solve Eq. (3.4) to find

$$C(1, 1] = \begin{bmatrix} -5 \\ -10 \end{bmatrix},$$

so

$$C = \begin{bmatrix} -5 & \frac{9}{5} & -\frac{3}{5} \\ -5 & 6 & \frac{7}{6} \\ -10 & 6 & -3 \end{bmatrix}.$$

Performing similar operations for the second matrix of level 1, we leave level 1 with the two 3×3 matrices

$$\begin{bmatrix} -5 & \frac{9}{5} & -\frac{3}{5} \\ -5 & 6 & \frac{7}{6} \\ -10 & 6 & -3 \end{bmatrix}, \quad \begin{bmatrix} -8 & \frac{9}{4} & -\frac{5}{8} \\ -8 & \frac{3}{2} & \frac{25}{3} \\ -\frac{24}{5} & \frac{3}{2} & -\frac{7}{3} \end{bmatrix}.$$

Level 0

Given $pivot = B[1] = pm_1 = -6$ and

$$L = B(1) = \begin{bmatrix} -5 & \frac{9}{5} & -\frac{3}{5} \\ -5 & 6 & \frac{7}{6} \\ -10 & 6 & -3 \end{bmatrix}, \quad R = B/B[1] = \begin{bmatrix} -8 & \frac{9}{4} & -\frac{5}{8} \\ -8 & \frac{3}{2} & \frac{25}{3} \\ -\frac{24}{5} & \frac{3}{2} & -\frac{7}{3} \end{bmatrix},$$

we desire to find the 4×4 matrix B of the form

$$B = \begin{bmatrix} -6 & * & * & * \\ * & -5 & \frac{9}{5} & -\frac{3}{5} \\ * & -5 & 6 & \frac{7}{6} \\ * & -10 & 6 & -3 \end{bmatrix}.$$

As in the previous level, $\operatorname{rank}(L - R) \neq 1$, so we find r, s, t, such that Eq. (3.8) is satisfied. Solving Eq. (3.7) for each 2×2 submatrix of $L - R$, we obtain two quadratic solutions for r, s and t:

$$r_1 = 20/7, \quad r_2 = 10, \quad s_1 = 5/2, \quad s_2 = 5/16, \quad t_1 = 25/36, \quad t_2 = 25/8.$$

Then, comparing Eq. (3.9) for each of the 8 possible combinations, we find that only $r = r_2$, $s = s_2$, $t = t_2$ makes $\operatorname{rank}(L - R) = 1$. Applying this r, s, and t to R we obtain a $\hat{R}$ which is dot similar to R:

$$\hat{R} = \begin{bmatrix} -8 & \frac{9}{4s} & -\frac{5}{8t} \\ -8s & \frac{3}{2} & \frac{25}{3r} \\ -\frac{24t}{5} & \frac{3r}{2} & -\frac{7}{3} \end{bmatrix} = \begin{bmatrix} -8 & \frac{36}{5} & -\frac{1}{5} \\ -\frac{5}{2} & \frac{3}{2} & \frac{5}{6} \\ -15 & 15 & -\frac{7}{3} \end{bmatrix}.$$

In this case, $\hat{R} = DRD^{-1}$ with

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{5}{16} & 0 \\ 0 & 0 & \frac{25}{8} \end{bmatrix},$$

and no transpose is necessary.

Now,

$$L - \hat{R} = \begin{bmatrix} -5 & \frac{9}{5} & -\frac{3}{5} \\ -5 & 6 & \frac{7}{6} \\ -10 & 6 & -3 \end{bmatrix} - \begin{bmatrix} -8 & \frac{36}{5} & -\frac{1}{5} \\ -\frac{5}{2} & \frac{3}{2} & \frac{5}{6} \\ -15 & 15 & -\frac{7}{3} \end{bmatrix} = \begin{bmatrix} 3 & -\frac{27}{5} & -\frac{2}{5} \\ -\frac{5}{2} & \frac{9}{2} & \frac{1}{3} \\ 5 & -9 & -\frac{2}{3} \end{bmatrix}.$$

Since 9/2 of row 2 is the maximum absolute diagonal entry of $L - \hat{R}$, we let $B[1, 1) = (L - \hat{R})[2, \{1, 2, 3\}] = [-5/2, 9/2, 1/3]$ and by Eq. (3.4) we find

$$B(1, 1] = \begin{bmatrix} \frac{36}{5} \\ -6 \\ 12 \end{bmatrix},$$

so

$$B = \begin{bmatrix} -6 & -\frac{5}{2} & \frac{9}{2} & \frac{1}{3} \\ \frac{36}{5} & -5 & \frac{9}{5} & -\frac{3}{5} \\ -6 & -5 & 6 & \frac{7}{6} \\ 12 & -10 & 6 & -3 \end{bmatrix}.$$

Of course B is not equal to the original matrix A, but $A = DBD^{-1}$ where

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\frac{5}{6} & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{12} \end{bmatrix}.$$

For simplicity of illustration, the deskewing of Section 3.2.5 has not been applied to the output matrix B.

Example 4.2. To show how zero principal minors are handled as discussed in Section 3.2.4, we consider running PM2MAT on the principal minors of the effectively ODF matrix

$$A = \begin{bmatrix} 2 & 2 & 5 \\ 2 & 2 & -3 \\ 7 & 3 & -1 \end{bmatrix}.$$

Using the convention of Example 4.1, A has principal minors:

$$pm = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ [1] & [2] & [1,2] & [3] & [1,3] & [2,3] & [1,2,3] \\ 2 & 2 & 0 & -1 & -37 & 7 & -64 \end{pmatrix}.$$

The 1×1 matrices of level 2 consist of the ratios

$$[pm_4], \quad [pm_5/pm_1], \quad [pm_6/pm_2], \quad [pm_7/pm_3]. \qquad \text{(Level 2)}$$

Since $pm_3 = 0$, building these matrices would fail if the zero principal minor code were not invoked.

The goal of zero principal minor loop at the beginning of PM2MAT is to create a consistent (taking into account the modified pivot entries) set of principal minors where the initial $2^{n-1} - 1$ principal minors are nonzero. It does this by making all the pivot or (1,1) entries of the matrices that correspond to zero principal minors have pivots equal to *ppivot* instead of zero.

In the example at hand, we will take $ppivot = 1$ for simplicity. Since pm_3 is computed by knowing that the pivot entry of the right matrix of level 1 is pm_3/pm_1, we assign $pm_3 = ppivot \cdot pm_1 = 1 \cdot 2 = 2$. Of course, in general, changing a single principal minor does not even result in a set of principal minors that are consistent, so the next part of the zero principal minor loop is devoted to computing the sums of principal minors that follow from applying the multilinearity of the determinant (also see the discussion of zero pivots in [2]) to the change just made. Since pm_3 is found from the (1,1) entry of $A/A[1]$, only principal minors from descendants (in the top to bottom sense of the tree of Fig. 1) of the Schur complement of $A/A[1]$ need to change. The only principal minor that satisfies this condition is pm_7. Thus, the remaining required change to the vector of principal minors is to let $pm_7 = pm_7 + ppivot \cdot pm_5 = -64 + (-37) = -101$. The final set of principal minors that we enter the main loop of PM2MAT is then

$$nzpm = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ [1] & [2] & [1,2] & [3] & [1,3] & [2,3] & [1,2,3] \\ 2 & 2 & 2 & -1 & -37 & 7 & -101 \end{pmatrix},$$

where *nzpm* signifies "nonzero principal minors". The first two levels of PM2MAT then yield

$$[-1], \quad [-37/2], \quad [7/2], \quad [-101/2], \qquad \text{(Level 2)}$$

$$L = \begin{bmatrix} 2 & -\frac{9}{2} \\ 2 & -1 \end{bmatrix}, \quad nzR = \begin{bmatrix} ppivot = 1 & 32 \\ 1 & -\frac{37}{2} \end{bmatrix}. \qquad \text{(Level 1)}$$

At this point we recall that $nzpm$ was designed to create $ppivot = 1$ in the (1, 1) entry of the matrix nzR, but the original principal minors corresponded to the (1, 1) entry of nzR being zero. Subtracting $ppivot = 1$ from $nzR[1]$ and continuing we obtain:

$$L = \begin{bmatrix} 2 & -\frac{9}{2} \\ 2 & -1 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 32 \\ 1 & -\frac{37}{2} \end{bmatrix}, \tag{Level 1}$$

$$B = \begin{bmatrix} 2 & -\frac{10}{3} & \frac{35}{2} \\ -\frac{6}{5} & 2 & -\frac{9}{2} \\ 2 & 2 & -1 \end{bmatrix}. \tag{Level 0}$$

Note that B is diagonally similar to A with transpose, and thus has the principal minors pm. If we had continued to run PM2MAT with the level 1 matrix nzR, we would have obtained a matrix that had the principal minors $nzpm$ instead.

Example 4.3. To show that PM2MAT may succeed in producing an output matrix which is not diagonally similar with transpose to another matrix having the same principal minors, consider running PM2MAT on the principal minors of

$$A = \begin{bmatrix} 2 & 4 & 1 & 1 \\ 3 & 5 & 1 & 1 \\ 1 & 1 & 7 & 13 \\ 1 & 1 & 6 & -3 \end{bmatrix},$$

which are

$$pm = [2, 5, -2, 7, 13, 34, -14, -3, -7, -16, 6, -99, -183, -480, 198].$$

This matrix is not ODF by (c) of Definition 2.3, but it is structured so that regardless of which dot similarity is chosen to make the final 3×3 difference of L and R rank 1, a matrix with the same principal minors results. By chance, PM2MAT returns the matrix

$$B = \begin{bmatrix} 2 & 6 & \frac{2}{5} & \frac{12}{35} \\ 2 & 5 & \frac{1}{5} & \frac{6}{35} \\ \frac{5}{2} & 5 & 7 & \frac{78}{7} \\ \frac{35}{12} & \frac{35}{6} & 7 & -3 \end{bmatrix}$$

(before deskewing) which is not diagonally similar with transpose to A. Note that in this example, the output of PM2MAT is dependent on small round-off errors which may vary from platform to platform, although the principal minors of the output matrix will be virtually the same regardless.

5. A polynomial time algorithm

As the size n of the matrix $A \in \mathscr{M}_n(\mathbb{C})$ increases, A has many more principal minors than entries. We thus know that these principal minors are dependent on each other. To study some of these dependencies, we prove the following preliminary result.

Lemma 5.1. *Let $A \in \mathscr{M}_n(\mathbb{C})$, $n \geqslant 4$, have $2^n - 1$ principal minors $\{pm_1, pm_2, \dots, pm_{2^n-1}\}$. If A is* ODF, *then pm_{2^n-1} is determined by $\{pm_1, pm_2, \dots, pm_{2^n-2}\}$.*

Proof. Let $\{pm_1, pm_2, \dots, pm_{2^n-2}, \hat{pm}_{2^n-1}\}$ be passed to PM2MAT where $\hat{pm}_{2^n-1} \neq pm_{2^n-1}$ and for convenience $\hat{pm}_{2^n-1} \neq 0$. The proof proceeds by showing that the 4×4 matrices of level $n-4$ using this set of principal minors with the last principal minor modified will be identical to those 4×4 matrices of level $n-4$ we would have obtained using the original principal minors $\{pm_1, pm_2, \dots, pm_{2^n-2}, pm_{2^n-1}\}$.

The queue of 1×1 matrices of level $n-1$ will be identical to the queue of 1×1 matrices of level $n-1$ using the original principal minors $\{pm_1, pm_2, \dots, pm_{2^n-2}, pm_{2^n-1}\}$ except for the last value. Therefore, the queue of 2×2 matrices of level $n-2$ will be identical to the same queue using the original principal minors except for the final matrix, whose (1,2) entry will be different. Let

$$R = \begin{bmatrix} r_{11} & \hat{r_{12}} \\ r_{21} & r_{22} \end{bmatrix}$$

be the last matrix of level $n-2$. Let B be the last matrix of level $n-3$ we desire to produce from level $n-2$ and as usual we have $R = B/B[1]$ and $L = B(1)$. When we solve the quadratic Eq. (3.7) and modify R so that $\operatorname{rank}(L-R) = 1$, we have

$$C = L - R = \begin{bmatrix} c_{11} & \hat{c_{12}} \\ \hat{c_{21}} & c_{22} \end{bmatrix},$$

where only $\hat{c_{12}}$ and $\hat{c_{21}}$ are different from entries we would have obtained using the original principal minors. Note, however, that $\operatorname{rank}(L-R) = \operatorname{rank}(C) = 1$ implies that $\det(C) = 0$, so in fact $\hat{c_{12}}\hat{c_{21}} = c_{12}c_{21}$. Hence, C is diagonally similar to the matrix we would have obtained using the original principal minors. Although B will be different from the B we would have obtained with the original principal minors, it will only be different in its (1,3) and (3,1) entries (or (1,2) and (2,1) entries, depending on the relative magnitudes of c_{11} and c_{22} by Eqs. (3.3), (3.4) and (3.5)), while the product of these entries will be the same. Since A is ODF and B is dot similar to the corresponding Schur complement of A, there will only be one dot similarity with B that will make the difference of B with its corresponding left matrix rank 1 (see (c) of Definition 2.3). Therefore, all the 4×4 matrices of level $n-4$ will be identical to those that were obtained using the original principal minors. □

Example 5.2. To illustrate the above lemma, let us revisit Example 4.1 from Section 4. The operation of PM2MAT is summarized as follows:

$$pm = [-6, -5, 48, 6, -9, -21, -36, -3, 14, 9, -94, -25, 96, 59, 6]$$

$$[-3], \quad \left[-\tfrac{7}{3}\right], \quad \left[-\tfrac{9}{5}\right], \quad \left[-\tfrac{47}{24}\right], \quad \left[-\tfrac{25}{6}\right], \quad \left[-\tfrac{32}{3}\right], \quad \left[-\tfrac{59}{21}\right], \quad \left[-\tfrac{1}{6}\right] \qquad \text{(Level 3)}$$

$$\begin{bmatrix} 6 & \frac{7}{6} \\ 6 & -3 \end{bmatrix}, \quad \begin{bmatrix} \frac{3}{2} & \frac{25}{3} \\ \frac{3}{2} & -\frac{7}{3} \end{bmatrix}, \quad \begin{bmatrix} \frac{21}{5} & \frac{106}{105} \\ \frac{21}{5} & -\frac{9}{5} \end{bmatrix}, \quad \begin{bmatrix} -\frac{3}{4} & -\frac{43}{24} \\ -\frac{3}{4} & -\frac{47}{24} \end{bmatrix} \qquad \text{(Level 2)}$$

$$\begin{bmatrix} -5 & \frac{9}{5} & -\frac{3}{5} \\ -5 & 6 & \frac{7}{6} \\ -10 & 6 & -3 \end{bmatrix}, \quad \begin{bmatrix} -8 & \frac{9}{4} & \frac{15}{2} \\ -8 & \frac{3}{2} & \frac{25}{3} \\ \frac{2}{5} & \frac{3}{2} & -\frac{7}{3} \end{bmatrix} \qquad \text{(Level 1)}$$

$$\begin{bmatrix} -6 & -\frac{5}{2} & \frac{9}{2} & \frac{1}{3} \\ \frac{36}{5} & -5 & \frac{9}{5} & -\frac{3}{5} \\ -6 & -5 & 6 & \frac{7}{6} \\ 12 & -10 & 6 & -3 \end{bmatrix} \qquad \text{(Level 0)}$$

If instead we use the input (entries that are different are in **bold**)

$$pm = [-6, -5, 48, 6, -9, -21, -36, -3, 14, 9, -94, -25, 96, 59, \mathbf{1}]$$

we obtain

$$[-3], \quad \left[-\tfrac{7}{3}\right], \quad \left[-\tfrac{9}{5}\right], \quad \left[-\tfrac{47}{24}\right], \quad \left[-\tfrac{25}{6}\right], \quad \left[-\tfrac{32}{3}\right], \quad \left[-\tfrac{59}{21}\right], \quad \left[-\tfrac{\mathbf{1}}{\mathbf{36}}\right] \qquad \text{(Level 3)}$$

$$\begin{bmatrix} 6 & \frac{7}{6} \\ 6 & -3 \end{bmatrix}, \quad \begin{bmatrix} \frac{3}{2} & \frac{25}{3} \\ \frac{3}{2} & -\frac{7}{3} \end{bmatrix}, \quad \begin{bmatrix} \frac{21}{5} & \frac{106}{105} \\ \frac{21}{5} & -\frac{9}{5} \end{bmatrix}, \quad \begin{bmatrix} -\frac{3}{4} & -\frac{\mathbf{139}}{\mathbf{72}} \\ -\frac{3}{4} & -\frac{47}{24} \end{bmatrix} \qquad \text{(Level 2)}$$

$$\begin{bmatrix} -5 & \frac{9}{5} & -\frac{3}{5} \\ -5 & 6 & \frac{7}{6} \\ -10 & 6 & -3 \end{bmatrix}, \quad \begin{bmatrix} -8 & \frac{9}{4} & -0.6304\ldots \\ -8 & \frac{3}{2} & \frac{25}{3} \\ \mathbf{-4.7590}\ldots & \frac{3}{2} & -\frac{7}{3} \end{bmatrix} \qquad \text{(Level 1)}$$

$$\begin{bmatrix} -6 & -\frac{5}{2} & \frac{9}{2} & \frac{1}{3} \\ \frac{36}{5} & -5 & \frac{9}{5} & -\frac{3}{5} \\ -6 & -5 & 6 & \frac{7}{6} \\ 12 & -10 & 6 & -3 \end{bmatrix} \qquad \text{(Level 0)}$$

Rational expressions for the (1, 3) and (3, 1) entries of the last matrix of level 1 do not exist, but note that $(15/2) \cdot (2/5) = 3 \approx (-0.6304) \cdot (-4.7590)$.

As a consequence to Lemma 5.1 we observe that generically all the "large" principal minors of a matrix can be expressed in terms of the "small" principal minors.

Theorem 5.3. *Let $A \in \mathscr{M}_n(\mathbb{C})$, $n \geqslant 4$. If A is* ODF, *each principal minor which is indexed by a set with* 4 *or more elements can be expressed in terms of the principal minors which are indexed by* 3 *or fewer elements.*

Proof. Consider all the submatrices of A indexed by exactly 4 elements. Applying Lemma 5.1 to each of them, we see that all of the principal minors indexed by exactly 4 elements are functions of all the smaller principal minors (the determinants of all 1×1, 2×2 and 3×3 principal submatrices of A). Applying Lemma 5.1 to each principal minor of A indexed by exactly 5 elements, these are functions of all of the principal minors indexed by 4 or fewer elements. However, all the principal minors with exactly 4 elements are functions of the principal minors indexed by 3 or fewer elements, so all the principal minors indexed by 5 elements are functions of the principal minors indexed by 3 or fewer elements. The result follows by induction. $\square$

Theorem 5.3 motivates a "fast" version of PM2MAT that only takes as inputs the 1×1, 2×2 and 3×3 principal minors but produces the same output matrix as PM2MAT if the input principal minors come from a ODF matrix. To this end, a matching, fast (polynomial time) version of MAT2PM was written to produce only these small principal minors and is described first.

FMAT2PM
or a ODF matrix of size n, there are only

$$\binom{n}{1} + \binom{n}{2} + \binom{n}{3} = n + \frac{n(n-1)}{2} + \frac{n(n-1)(n-2)}{6} = \frac{n(n^2+5)}{6}$$

principal minors with index sets of size $\leqslant 3$ on which all the other principal minors depend. A fast and limited version of MAT2PM was implemented that produces only these principal minors in the function FMAT2PM in Appendix C. Instead of having 2^k matrices at each level which yield 2^k principal minors at $level = k$, there are only

$$\binom{k}{2} + \binom{k}{1} + \binom{k}{0} = \frac{k(k+1)}{2} + 1 \tag{5.13}$$

matrices that need be computed resulting in the corresponding number of principal minors being produced. This is because at $level = k$ we only need to compute those principal minors involving the index $k+1$ and 2 smaller indices to produce the principal minors which come from 3×3 submatrices. The other two terms of (5.13) are similarly derived: the $k+1$ index and 1 smaller index is used to compute the principal minors which come from 2×2 submatrices, and each level has one new minor from the 1×1 submatrix which is referenced by the index $k+1$.

From (5.13), it is easily verified that at each $level = k$, precisely k more matrices are produced than were computed at $level = k-1$.

Simple logic can be used to determine which matrices in the input queue need to be processed to produce matrices in the output queue. If the (1,1) entry of a matrix corresponds to a principal minor with an index set of 2 or less (a 2×2 or 1×1 principal minor) then we take the submatrix and Schur complement of the matrix, just as we do in MAT2PM since this matrix will have descendants which correspond to principal minors from 3×3 submatrices. However, if the (1, 1) entry of a matrix corresponds to a principal minor with an index set of 3, then we only take the submatrix of this matrix; no Schur complement is computed or stored. This scheme prevents matrices that would be used to find principal minors with an index set of 4 or more from ever being computed. To keep track the index sets of the principal minors, the index of the principal minor as it would have been computed in MAT2PM is stored in the vector of indices $pmidx$. In the code, the MAT2PM indices in $pmidx$ are referred to as the *long* indices of a principal minor, while the indices in the pm vector produced by FMAT2PM are called the *short* indices of a given principal minor. The vector of long indices $pmidx$ is output along with the vector of principal minors in pm by FMAT2PM.

At each level, the indices pertaining to the principal minors which are computed at that level are put in $pmidx$. The following table shows how $pmidx$ is incrementally computed at each level:

$level$	$pmidx$
0	$[1, 0, 0, \dots, 0]$
1	$[1, 2, 3, 0, 0, \dots, 0]$
2	$[1, 2, 3, 4, 5, 6, 7, 0, \dots, 0]$
3	$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, \dots, 0]$
4	$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17,$ $18, 19, 20, 21, 22, 24, 25, 26, 28, 0, \dots, 0]$
$\vdots$	$\vdots$

As an aid to interpreting the values in $pmidx$, the utilities IDX2V and V2IDX in Appendix F and Appendix E respectively have been included. IDX2V takes a long index in $pmidx$ and produces the index set of the submatrix that corresponds to it and returns this as a vector. V2IDX is the inverse of this operation.

We note that FMAT2PM could be adapted analogously to compute all principal minors up to any fixed size of index set.

FPM2MAT

Given *pm* and $pmidx$ in the format produced by FMAT2PM, FPM2MAT can invert only these 1×1, 2×2 and 3×3 principal minors into a matrix that is unique up to diagonal similarity with transpose if the principal minors come from a matrix which is ODF as a consequence of Theorem 5.3.

It is not strictly necessary that $pmidx$ be an input to FPM2MAT since it is fixed for any given matrix size n, but since $pmidx$ is produced naturally by FMAT2PM and since $pmidx$ can be thought of with *pm* as implementing a very sparse vector format, *pm* and $pmidx$ are input together to FPM2MAT.

Since FPM2MAT processes levels from the largest (bottom) to smallest (top), it was useful to make the vector $ipmlevels$ at the beginning of FPM2MAT which contains the short index of the beginning of each level. Then, at each level, FPM2MAT computes those matrices which have a submatrix (or left matrix L). If a Schur complement (an R matrix) exists for the matrix to be produced, this is input to *invschurc* just as in PM2MAT. Otherwise, a matrix of ones is passed in for the Schur complement to *invschurc*, since the particular values will prove to be of no consequence (see Example 5.2). Although this is slightly slower than just embedding L in a larger matrix, it has been found to produce more accurate results for large n.

The bulk of the operations of FPM2MAT including the subroutines *invschurc*, *solveright* and $deskew$ are unchanged from PM2MAT.

5.1. Notes on the operation of the fast versions

- FMAT2PM processes a 53×53 matrix in about the same time that MAT2PM can process a 20×20 matrix.
- FPM2MAT can invert the 24857 "small" principal minors of a 53×53 real or complex matrix into a matrix in $\mathscr{M}_{53}$ in a few minutes.
- Zero principal minors are not presently handled by either FMAT2PM or FPM2MAT for clarity of presentation, but the multilinearity of the determinant could be exploited in these programs to handle zero principal minors as is implemented in MAT2PM and PM2MAT.
- For FMAT2PM, only $k + 1$ matrices need to have Schur complements taken at each $level = k$. Thus, FMAT2PM is of order $O(n^4)$ since

$$\sum_{k=0}^{n-2}(k+1)2(n-(k+1))^2 = \frac{1}{6}(n^4-n^2),$$

where there are generally 2 operations for each element of the Schur complement. The scaling of the outer product by the inverse of the appropriate principal minor only affects $(n-(k+1))$ elements at level k.

- It has been found that for numerical reasons it is desirable to call *invschurc* with a dummy matrix of ones whenever the R Schur complement matrix is missing.
- To find the approximate operation count for FPM2MAT, we note that the quadratics (3.7) need to be solved for each upper triangular entry of each of the $k(k+1)/2+1$ matrices (see (5.13)) at each $level = k$. As in (3.11) we let q be the number of operations to solve the quadratic equation to find

$$\sum_{k=0}^{n-3}(k(k+1)/2+1)(n-(k+1))(n-(k+2))q/2$$
$$= \frac{q}{120}(n^5-5n^4+25n^3-55n^2+34n).$$

Similarly, letting p be the number of operations to evaluate equations of the form (3.9),

$$p\sum_{k=0}^{n-4}(k(k+1)/2+1)(8+4(n-(k+4))+(n-(k+3))(n-(k+4)))$$
$$= \frac{p}{60}(n^5+5n^4+45n^3-295n^2+974n-1320),$$

as in the sum of (3.12). Thus, FPM2MAT is $O(n^5)$.

6. Some consequences and remarks

In this section we will touch upon some subjects and problems related to PMAP and PM2MAT in the form of remarks.

Remark 6.1. With PM2MAT we can at least partially answer a question posed by Holtz and Schneider in [3, problem (4) pp. 265–266]. The authors pose PMAP as one step toward the inverse problem of finding a GKK matrix with prescribed principal minors.

Given a set of principal minors, we can indeed determine via PM2MAT whether or not there is an effectively ODF matrix A that has them as its principal minors. To accomplish this, we simply run PM2MAT on the given set of principal minors, producing a matrix A. Then, we run MAT2PM on this matrix to see if the matrix A has the desired principal minors. If the principal minors are consistent and belong to an effectively ODF matrix, the principal minors will match, up to numerical limitations. If the principal minors cannot be realized by a matrix or if the principal minors do not belong to an effectively ODF matrix, then the principal minors of A will not match the input principal minors. In the former case, the generalized Hadamard–Fischer inequalities and the Gantmacher–Krein–Carlson theorem (see [3]) can be used to decide whether there is a GKK and effectively ODF matrix with the given principal minors.

Remark 6.2. In Stouffer [5], it is claimed that for $A \in \mathscr{M}_n(\mathbb{C})$ there are only $n^2 - n + 1$ "independent" principal minors that all the other principal minors depend on, and as the first example of such a "complete" set the following set is offered:

$$S = \{A[i] : i \in \langle n \rangle\} \cup \{A[i, j] : i < j; i, j \in \langle n \rangle\} \cup \{A[1, i, j] : i < j; i, j \in \{2, 3, \ldots, n\}\}.$$

Indeed, S contains

$$\binom{n}{1} + \binom{n}{2} + \binom{n-1}{2} = n^2 - n + 1$$

principal minors. However, these principal minors are not independent since equations [5, (5) and (6), p. 358] may not even have solutions if the principal minors are not consistent or come from a matrix which is not ODF. Moreover, even generically, S does not determine A up to diagonal similarity with transpose for $n \geqslant 4$ since, in general, the last two equations of [5, (6), p. 358] have two solutions for each a_{ij}, a_{ji} pair. The set S in [5], however, does generically determine the other principal minors of A up to a finite set of possibilities.

Remark 6.3. The conditions of Definition 2.3 are difficult to verify from an input set of principal minors from an unknown source without actually running the PM2MAT algorithm on them. In particular, part (c) is difficult to verify even if a matrix that has the same principal minors as the input principal minors happens to be known. However, usually PM2MAT will print a warning if the conditions of the definition are not satisfied, and the strategy of Section 3.4 can be employed to quickly verify if PM2MAT has performed the desired task.

Remark 6.4. Note that Theorem 5.3 does not imply that the principal minors indexed by 3 or fewer elements are completely free. There are additional constraints among the smaller principal minors. If one chooses a set of principal minors indexed by 3 or fewer elements at random and uses FPM2MAT to invert these into a matrix A, the matrix A will not in general have the same principal minors indexed by 3 or fewer elements as the input principal minors.

Remark 6.5. If a random vector of principal minors is passed to PM2MAT, the resulting matrix will have all the desired 1×1 and 2×2 principal minors, and some of the 3×3 principal minors will be realized as well. However, none of the larger principal minors will be the same as those that were in the input random vector of principal minors.

7. Future work

Here we summarize some of our main goals for the future regarding PMAP and PM2MAT.

- Incorporate into PMAP and its solution algorithm structural and qualitative requirements of the matrix solutions. For example, we may want to seek solutions of special structure (e.g., positive definite, Hessenberg, Toeplitz) or having other special attributes (e.g., specified directed graph, entrywise nonnegativity).
- Expand the functionality of PM2MAT from effectively ODF matrices to a broader class of matrices.
- Identify applications of PMAP in theoretical and applied mathematics, and contribute to the solution of relevant problems.

- Examine the potential consequences and usage of our algorithmic approach to the P-problem. This the problem of checking whether a given matrix is a P-matrix or not. Recall that a P-matrix is one all of whose principal minors are positive. The P-problem is known to be an NP-hard problem [1].
  To be more specific, note all the principal minors that FMAT2PM computes being positive provides a polynomial time necessary condition for a matrix to be a P-matrix. If the positivity of the larger minors could be predicted from these 1×1, 2×2 and 3×3 principle minors for ODF matrices, then it would be possible to realize a polynomial time algorithm for the P-problem applied to this class of matrices.

Acknowledgments

The authors are grateful to an anonymous referee for a careful examination of the first draft of this paper leading to the introduction of condition (c) of Definition 2.3 and for several other improvements to the manuscript.

Appendix A. PM2MAT

```
% PM2MAT Finds a real or complex matrix that has PM as its principal
% minors.
%
%   PM2MAT produces a matrix with some, but perhaps not all, of its
%   principal minors in common with PM.  If the principal minors are
%   not consistent or the matrix that has them is not ODF, PM2MAT will
%   produce a matrix with different principal minors without warning.
%   Run MAT2PM on the output matrix A as needed.
%
%   A = PM2MAT(PM)
%  where PM is a 2^n - 1 vector of principal minors and A will be an n x n
%  matrix.
%
%   The structure of PM, where |A[v]| is the principal minor of "A" indexed
%   by the vector v:
%   PM: |A[1]| |A[2]| |A[1 2]| |A[3]| |A[1 3]| |A[2 3]| |A[1 2 3]| ...
function A = pm2mat(pm)
myeps = 1e-10;
n = log2(length(pm)+1);
% Make first (smallest) entry of zeropivs an impossible index
zeropivs = 0;
% Pick a random pseudo-pivot value that minimizes the chances that
% pseudo-pivoting will create a non-ODF matrix.
ppivot = 1.9501292851471754e+000;
% initialize globals to allow warnings to be printed only once
global WARN_A WARN_C WARN_I
WARN_A = false; WARN_C = false; WARN_I = false;
```

```
% To avoid division by zero, do an operation analogous to the zeropivot
% loop in mat2pm.
for i = 1:((length(pm)+1)/2 - 1)
    if (abs(pm(i)) < myeps)
        mask = i;
        zeropivs = union(zeropivs, i);
        ipm1 = bitand(mask, bitcmp(msb(mask),48));
        if ipm1 == 0
            pm(mask) = pm(mask) + ppivot;
        else
            pm(mask) = (pm(mask)/pm(ipm1) + ppivot)*pm(ipm1);
        end
        delta = msb(mask);
        delta2 = 2*delta;
        for j = mask+delta2:delta2:2^n - 1
            pm(j) = pm(j) + ppivot*pm(j - delta);
        end
    end
end
zeropivsidx = length(zeropivs) - 1;
zeropivsmax = zeropivs(zeropivsidx+1);

% initial processing is special, no call to invschurc is necessary
nq = 2^(n-1);
q = zeros(1,1,nq);
ipm1 = nq;
ipm2 = 1;
for i = 1:nq
    if i == 1
        q(1,1,i) = pm(ipm1);
    else
        q(1,1,i) = pm(ipm1)/pm(ipm2);
        ipm2 = ipm2+1;
    end
    ipm1 = ipm1+1;
end

%
% Main 'level' loop
%
for level = n-2:-1:0        % for consistency with mat2pm levels
    [n1, n1, nq] = size(q);
    nq = nq/2;
    n1 = n1+1;
    % The output queue has half the number of matrices, each one larger in
    % row and col dimension
```

```
    qq = zeros(n1, n1, nq);
    ipm1 = 2*nq-1;
    ipm2 = nq-1;
    for i = nq:-1:1     % process matrices in reverse order for zeropivs
        if (i == 1)
            pivot = pm(ipm1);
        else
            pivot = pm(ipm1)/pm(ipm2);
            ipm2 = ipm2-1;
        end
        qq(:,:,i) = invschurc(pivot, q(:,:,i), q(:,:,i+nq));
        if zeropivsmax == ipm1
            qq(1,1,i) = qq(1,1,i) - ppivot;
            zeropivsmax = zeropivs(zeropivsidx);
            zeropivsidx = zeropivsidx - 1;
        end
        ipm1 = ipm1-1;
    end
    q = qq;
end
A = q(:,:,1);
A = deskew(A);

if WARN_A
    % ODF (a) not satisfied
    fprintf(2,...
'PM2MAT: off diagonal zeros found, solution suspect.\n');
end
if WARN_C
    fprintf(2,...
'PM2MAT: multiple solutions to make rank(L-R)=1, solution suspect.\n');
end
if WARN_I
    fprintf(2, ...
'PM2MAT: input principal minors may be inconsistent, solution suspect.\n');
end

%
% Suppose A is an (m+1) x (m+1) matrix such that
%
% pivot = A(1,1)
% L = A(2:m+1, 2:m+1)
% R = L - A(2:m+1,1)*A(1,2:m+1)/pivot = (the Schur's complement with
% respect to the pivot or A/A[1]).
%
% Then invschurc finds such an (m+1) x (m+1) matrix A (not necessarily
```

```
% unique) given the pivot (a scalar), and the m x m matrices L and R.
%
% If rank(L-R) is not 1, modifies R so that L-R is rank 1.
%
function A = invschurc(pivot, L, R)
global WARN_C WARN_I
myeps_i = 1e-3*norm(R,inf);   % make these relative to magnitude of R
myeps_c = 1e-9*norm(R,inf);
m = length(R);

% Try to make (L-R) rank 1
if m == 2
    [t1,t2] = solveright(L(1,1), L(1,2), L(2,1), L(2,2),...
        R(1,1), R(1,2), R(2,1), R(2,2));

    % This is arbitrary, take the first.
    t = t1;

    R(2,1) = R(2,1)*t;
    R(1,2) = R(1,2)/t;
elseif m >= 3
    % We start with the lower right hand 3x3 submatrix.  We have 3
    % parameters, each with two possible solutions.  Only 1 of the 8
    % possible solutions need give us a L-R which is rank 1.  We find the
    % right solution by brute force.
    i1 = m-2;
    i2 = m-1;
    i3 = m;
    [r1,r2] = solveright(L(i2,i2), L(i2,i3), L(i3,i2), L(i3,i3),...
        R(i2,i2), R(i2,i3), R(i3,i2), R(i3,i3));
    [s1,s2] = solveright(L(i1,i1), L(i1,i2), L(i2,i1), L(i2,i2),...
        R(i1,i1), R(i1,i2), R(i2,i1), R(i2,i2));
    [t1,t2] = solveright(L(i1,i1), L(i1,i3), L(i3,i1), L(i3,i3),...
        R(i1,i1), R(i1,i3), R(i3,i1), R(i3,i3));
    % Perform a parameterized "row reduction" on the first two rows of this
    % matrix and compute the absolute value of the (2,3) entry.  One of
    % them will be nearly zero.
    r111 = abs((L(i2,i3) - R(i2,i3)/r1)*(L(i1,i1) - R(i1,i1)) - ...
        (L(i2,i1) - R(i2,i1) * s1)*(L(i1,i3) - R(i1,i3)/t1));
    r112 = abs((L(i2,i3) - R(i2,i3)/r1)*(L(i1,i1) - R(i1,i1)) - ...
        (L(i2,i1) - R(i2,i1)*s1)*(L(i1,i3) - R(i1,i3)/t2));
    r121 = abs((L(i2,i3) - R(i2,i3)/r1)*(L(i1,i1) - R(i1,i1)) - ...
        (L(i2,i1) - R(i2,i1)*s2)*(L(i1,i3) - R(i1,i3)/t1));
    r122 = abs((L(i2,i3) - R(i2,i3)/r1)*(L(i1,i1) - R(i1,i1)) - ...
        (L(i2,i1) - R(i2,i1)*s2)*(L(i1,i3) - R(i1,i3)/t2));
    r211 = abs((L(i2,i3) - R(i2,i3)/r2)*(L(i1,i1) - R(i1,i1)) - ...
```

```
    (L(i2,i1) - R(i2,i1)*s1)*(L(i1,i3) - R(i1,i3)/t1));
r212 = abs((L(i2,i3) - R(i2,i3)/r2)*(L(i1,i1) - R(i1,i1)) - ...
    (L(i2,i1) - R(i2,i1)*s1)*(L(i1,i3) - R(i1,i3)/t2));
r221 = abs((L(i2,i3) - R(i2,i3)/r2)*(L(i1,i1) - R(i1,i1)) - ...
    (L(i2,i1) - R(i2,i1)*s2)*(L(i1,i3) - R(i1,i3)/t1));
r222 = abs((L(i2,i3) - R(i2,i3)/r2)*(L(i1,i1) - R(i1,i1)) - ...
    (L(i2,i1) - R(i2,i1)*s2)*(L(i1,i3) - R(i1,i3)/t2));
rv = [r111, r112, r121, r122, r211, r212, r221, r222];
mn = min(rv);
if (r111 == mn)
    r = r1; s = s1; t = t1;
elseif (r112 == mn)
    r = r1; s = s1; t = t2;
elseif (r121 == mn)
    r = r1; s = s2; t = t1;
elseif (r122 == mn)
    r = r1; s = s2; t = t2;
elseif (r211 == mn)
    r = r2; s = s1; t = t1;
elseif (r212 == mn)
    r = r2; s = s1; t = t2;
elseif (r221 == mn)
    r = r2; s = s2; t = t1;
else % (r222 == mn)
    r = r2; s = s2; t = t2;
end
if mn > myeps_i
    WARN_I = true;
end
if sum(rv < myeps_c) > 1
    WARN_C = true;
end
R(i3,i2) = R(i3,i2)*r;
R(i2,i3) = R(i2,i3)/r;
R(i2,i1) = R(i2,i1)*s;
R(i1,i2) = R(i1,i2)/s;
R(i3,i1) = R(i3,i1)*t;
R(i1,i3) = R(i1,i3)/t;


% Now the lower right hand 3x3 submatrix of L-R has rank 1.  Then we
% fix up the rest of L-R.
for i1 = m-3:-1:1
    i2 = i1+1;
    i3 = i1+2;
```

```
% Now the inside lower right submatrix is done, so we
% only have 2 free parameters and 4 combinations to examine.
[s1,s2] = solveright(L(i1,i1), L(i1,i2), L(i2,i1), L(i2,i2),...
    R(i1,i1), R(i1,i2), R(i2,i1), R(i2,i2));
[t1,t2] = solveright(L(i1,i1), L(i1,i3), L(i3,i1), L(i3,i3),...
    R(i1,i1), R(i1,i3), R(i3,i1), R(i3,i3));

r11 = abs((L(i2,i3) - R(i2,i3))*(L(i1,i1) - R(i1,i1)) - ...
    (L(i2,i1) - R(i2,i1)*s1)*(L(i1,i3) - R(i1,i3)/t1));
r12 = abs((L(i2,i3) - R(i2,i3))*(L(i1,i1) - R(i1,i1)) - ...
    (L(i2,i1) - R(i2,i1)*s1)*(L(i1,i3) - R(i1,i3)/t2));
r21 = abs((L(i2,i3) - R(i2,i3))*(L(i1,i1) - R(i1,i1)) - ...
    (L(i2,i1) - R(i2,i1)*s2)*(L(i1,i3) - R(i1,i3)/t1));
r22 = abs((L(i2,i3) - R(i2,i3))*(L(i1,i1) - R(i1,i1)) - ...
    (L(i2,i1) - R(i2,i1)*s2)*(L(i1,i3) - R(i1,i3)/t2));
rv = [r11, r12, r21, r22];
mn = min(rv);
if (r11 == mn)
    s = s1; t = t1;
elseif (r12 == mn)
    s = s1; t = t2;
elseif (r21 == mn)
    s = s2; t = t1;
else % (r22 == mn)
    s = s2; t = t2;
end
if mn > myeps_i
    WARN_I = true;
end
if sum(rv < myeps_c) > 1
    WARN_C = true;
end
R(i2,i1) = R(i2,i1)*s;
R(i1,i2) = R(i1,i2)/s;
R(i3,i1) = R(i3,i1)*t;
R(i1,i3) = R(i1,i3)/t;
for j = i1+3:m
    % Finally, once the second row of the submatrix we are working
    % on is uniquely solved, we just pick the solution to the
    % quadratic such that the first row is a multiple of the
    % second row.  Note that one of r1, r2 will be almost zero.
    % Solving the quadratics leads to much better performance
    % numerically than just taking multiples of the second or
    % any other row.
    %
```

```
            j1 = i1+1;
            [t1,t2] = solveright(L(i1,i1), L(i1,j), L(j,i1), L(j,j),...
                R(i1,i1), R(i1,j), R(j,i1), R(j,j));
            r1 = abs((L(j1,j) - R(j1,j))*(L(i1,i1) - R(i1,i1)) - ...
                (L(j1,i1) - R(j1,i1))*(L(i1,j) - R(i1,j)/t1));
            r2 = abs((L(j1,j) - R(j1,j))*(L(i1,i1) - R(i1,i1)) - ...
                (L(j1,i1) - R(j1,i1))*(L(i1,j) - R(i1,j)/t2));
            if (r1 <= r2)
                t = t1;
            else
                t = t2;
            end
            rv = [r1, r2];
            if mn > myeps_i
                WARN_I = true;
            end
            if sum(rv < myeps_c) > 1
                WARN_C = true;
            end

           R(j,i1) = R(j,i1)*t;
            R(i1,j) = R(i1,j)/t;
        end
    end
end

B = (L-R);    % a rank 1 matrix
[mn, idxmax] = max(abs(diag(B)));
% For numerical reasons use the largest diagonal element as a base to find
% the two vectors whose outer product is B*pivot
yT = B(idxmax,:);
if yT(idxmax) == 0
    % This shouldn't happen normally, but to prevent
    % divide by zero when we set all "dependent" principal
    % minors (with index sets greater than or equal to a constant)
    % to the same value, let yT be something.
    yT = ones(1,m);
end
x = B(:,idxmax)*pivot / yT(idxmax);
A = zeros(m+1);
A(1,1) = pivot;
A(1,2:m+1) = yT;
A(2:m+1,1) = x;
A(2:m+1,2:m+1) = L;
```

```
%
% Returns the two possible real solutions that will make L-R rank one if we
% let
% r21 = r21*t? (where ? = 1 or 2) and
% r12 = r12/t?
%
function [t1,t2] = solveright(l11,l12,l21,l22,r11,r12,r21,r22)
global WARN_A
x1 = l11-r11;
x2 = l22-r22;
d = sqrt(x1^2*x2^2 + l12^2*l21^2 + r12^2*r21^2 - 2*x1*x2*l12*l21 - ...
    2*x1*x2*r12*r21-2*l12*l21*r21*r12);
if (l12 == 0)||(r21 == 0)
    % This shouldn't happen normally, but to prevent
    % divide by zero when we set all "dependent" principal
    % minors (with index sets greater than or equal to a constant)
    % to the same value, let [t1,t2] be something.
    t1 = 1;
    t2 = 1;
    WARN_A = true;
else
    t1 = (-x1*x2 + l12*l21 + r12*r21 - d)/(2*l12*r21);
    t2 = (-x1*x2 + l12*l21 + r12*r21 + d)/(2*l12*r21);

    % This also shouldn't happen.  Comment above applies.
    if (t1 == 0)||(t2 == 0)
        WARN_A = true;
        if (t1 == 0)&&(t2 == 0)
            t1 = 1;
            t2 = 1;
        elseif (t1 == 0)
            % return better solution in t1 for m=2 case in invschurc
            t1 = t2;
            t2 = 1;
        else  % (t2 == 0)
            t2 = 1;
        end
    end
end


%
% Makes abs(A(1,i)) = abs(A(i,1)) through diagonal similarity for all i.
%
function A = deskew(A)
n = length(A);
d = ones(n,1);
```

```
for i = 2:n
    if A(i,1) ~= 0  % don't divide by 0
        d(i) = sqrt(abs(A(1,i)/A(i,1)));
        if (d(i) > 1e6)||(d(i) < 1e-6)
            % something is likely wrong, use 1 instead
            d(i) = 1;
        end
    end     % else leave d(i) = 1
end

% If D = diag(d), this effectively computes A = D*A*inv(D)
for i = 2:n
    A(i,:) = A(i,:)*d(i);
end
for i = 2:n
    A(:,i) = A(:,i)/d(i);
end

% Returns the numerical value of the most significant bit of x.
% For example, msb(7) = 4, msb(6) = 4, msb(13) = 8.
function m = msb(x)
persistent MSBTABLE     % MSBTABLE persists between calls to mat2pm
if is empty(MSBTABLE)
    % If table is empty, initialize it
    MSBTABLE = zeros(255,1);
    for i=1:255
        MSBTABLE(i) = msbslow(i);
    end
end

m = 0;
% process 8 bits at a time for speed
if x ~= 0
    while x ~= 0
        x1 = x;
        x = bitshift(x, -8);    % 8 bit left shift
        m = m + 8;
    end
    m = bitshift(MSBTABLE(x1), m-8); % right shift
end

% Returns the numerical value of the most significant bit of x.
% For example, msb(7) = 4, msb(6) = 4, msb(13) = 8.  Slow version
% used to build a table.
function m = msbslow(x)
m = 0;
```

```
if x ~= 0
    m = 1;
    while  x ~= 0
        x = bitshift(x, -1);
        m = 2*m;
    end
    m = m/2;
end
```

Appendix B. PMFRONT

```
% PMFRONT  Finds a real or complex matrix that has PM as its principal
% minors if possible, prints out a warning if no such matrix can
% be found by PM2MAT.
%
%   A = PMFRONT(PM)
%   where PM is a 2^n - 1 vector of principal minors and A will be an n x n
%   matrix.
%
%   The structure of PM, where |A[v]| is the principal minor of "A" indexed
%   by the vector v:
%   PM: |A[1]| |A[2]| |A[1 2]| |A[3]| |A[1 3]| |A[2 3]| |A[1 2 3]| ...
function A = pmfront(pm)
myeps = 1e-5;   % tolerance for relative errors in the principal minors

% First run pm2mat
A = pm2mat(pm);

% Next, run mat2pm on the result
pm1 = mat2pm(A);

smallestpm = min(abs(pm));
if smallestpm < 1e-10
    fprintf(2, ...
'There are principal minors very close to zero, relative errors in\n');
    fprintf(2, ...
'principal minors may not be meaningful.  Consider the absolute error\n')
    fprintf(2, ...
'to decide if PM2MAT succeeded.\n');
    err = norm((pm-pm1),inf);

    fprintf(2, ...
'The maximum absolute error in the principal minors is %e\n', err);
else
    % Compare the results in terms of the relative error in the pm's
    err = norm((pm-pm1)./abs(pm),inf);
```

```
    if err > myeps
        fprintf(2, 'PM2MAT failed\n');
        fprintf(2, ...
'No matrix could be found that has all the requested principal minors.\n');
        fprintf(2, ...
'The PM''s are inconsistent or they come from a non-ODF matrix.\n');
    else
        fprintf(2, 'PM2MAT succeeded\n');
    end
    fprintf(2, ...
'The maximum relative error in the principal minors is %e\n', err);
end
```

Appendix C. FMAT2PM

```
% FMAT2PM Finds all 1x1, 2x2 and 3x3 principal minors of an n x n matrix.
%   [PM, PMIDX] = FMAT2PM(A)
%   where "A" is an n x n matrix (zero pivots not handled).
%   MAT2PM returns a vector of all the 1x1, 2x2, and 3x3 principal minors
%   of the matrix "A" in PM.  Also returns PMIDX, a vector of indices
%   that gives the index of the given principal minor in the full
%   binary ordered PM vector that MAT2PM produces.  Thus, for example,
%   if
%
%   A = rand(30);
%   [pm, pmidx] = fmat2pm(A);
%
%   then
%
%   det(A([25 29 30],[25 29 30]))
%
%   is the same as
%
%   pm(find(pmidx == v2idx([25 29 30])))
function [pm, pmidx] = fmat2pm(a)
% Only works on up to 53x53 matrices due to restrictions on indices.
n = length(a);


% nchoosek(n,1) + nchoosek(n,2) + nchoosek(n,3);
pm = zeros(1, (n^2 + 5)*n/6);     % where the principal minors are stored
pmidx = zeros(1, (n^2 + 5)*n/6);  % place to store full (mat2pm) indices
pmidx(1) = 1;
ipm = 1;                          % short (new) index for storing principal minors
q = zeros(n,n,1);                 % q is the input queue of unprocessed matrices
q(:,:,1) = a;                     % initial queue just has 1 matrix to process
```

```
%
% Main 'level' loop
%
for level = 0:n-1
    [n1, n1, nq] = size(q);
    nq2 = nq + level + 1;
    qq = zeros(n1-1, n1-1, nq2);
    ipmlevel = ipm + nq - 1;    % short index of beginning of the level
    ipm2 = 1;
    level2 = 2^level;
    for i = 1:nq
        a = q(:,:,i);
        pm(ipm) = a(1,1);
        ipm1 = pmidx(ipm);      % long index of current pm
        if n1 > 1
            if bitcount(ipm1) < 3
                b = a(2:n1,2:n1);
                % assume all pivots are non-zero
                d = a(2:n1,1)/pm(ipm);
                c = b - d*a(1,2:n1);

                qq(:,:,i) = b;
                pmidx(ipmlevel+i) = ipm1 + level2;
                qq(:,:,nq+ipm2) = c;
                pmidx(ipmlevel+nq+ipm2) = ipm1 + 2*level2;
                ipm2 = ipm2+1;
            else
                b = a(2:n1,2:n1);
                qq(:,:,i) = b;
                pmidx(ipmlevel+i) = ipm1 + level2;
            end
        end

        if i > 1
            pm(ipm) = pm(ipm)*pm(pmfind(pmidx, ipm1 - level2, ipmlevel));
        end
        ipm = ipm + 1;
    end
    q = qq;
end

% Returns the number of bits set in x, or 4 if more than
% 4 are set.
% For example, msb(7) = 3, msb(6) = 2, msb(10) = 2.
function c = bitcount(x)
c = 0;
while x ~= 0
```

```
    if bitand(x,1) == 1
        c = c + 1;
        if c >= 4
            return;     % no reason to keep counting
        end
    end
    x = bitshift(x, -1);    % shift right
end

%
% Find i0 in pmidx, returning its index, using a binary search,
% since pmidx is in ascending order.
%
% Same functionality as
%
% find(pmidx == i0)
%
% only faster.
%
function i = pmfind(pmidx, i0, n)
% 1:n is the part of pmidx that has values so far
iLo = 1;
iHi = n;
if pmidx(iHi) <= i0
    if pmidx(iHi) == i0
        i = n;
    else
        i = [];
    end
    return;
end
iOld = -1;
i = iLo;
while i ~= iOld
    iOld = i;
    i = floor((iHi + iLo)/2);
    if pmidx(i) < i0
        iLo = i;
    elseif pmidx(i) > i0
        iHi = i;
    else
        return;
    end
end

i = [];
return;
```

Appendix D. FPM2MAT

```
% FPM2MAT Finds a real or complex matrix that has PM as its 1x1, 2x2,
% and 3x3 principal minors if possible.
%
%   FPM2MAT produces a matrix with some, but perhaps not all, of its
%   principal minors in common with PM.  If the principal minors are
%   not consistent or the matrix that has them is not ODF, FPM2MAT will
%   produce a matrix with different principal minors without warning.
%   Run FMAT2PM on the output matrix A as needed.
%
%   A = FPM2MAT(PM, PMIDX)
%   where PM and PMIDX are in the format produced by FMAT2PM.
%
function [A] = fpm2mat(pm, pmidx)
% Only works on up to 53x53 matrices due to restrictions on indices.

% Since length(pm) = (n^3 + 5*n)/6, the computation below suffices to
% find n given length(pm).
n = floor((6*length(pm))^(1/3));

% initialize globals to allow warnings to be printed only once
global WARN_A WARN_C WARN_I
WARN_A = false; WARN_C = false; WARN_I = false;


% ipmlevels is a vector of the short indices of the start of each level
% (minus one for convenience of indexing)
ipmlevels = zeros(1,n);
ipmlevels(1) = 0;
for level = 1:n-1;
    ipmlevels(level+1) = ipmlevels(level) + level*(level-1)/2 + 1;
end

% no call to invschurc is necessary for level n-1
nq = n*(n-1)/2 + 1; q = zeros(1,1,nq);
ipm = ipmlevels(n) + 1;     % short index of current pm
level2 = 2^(n-1);
for i = 1:nq
    if i == 1
        q(1,1,i) = pm(ipm);

    else
        q(1,1,i) = pm(ipm)/pm(pmfind(pmidx, pmidx(ipm) - level2));
    end
```

```
    ipm = ipm+1;
end


%
% Main 'level' loop
%
for level = n-2:-1:0        % for consistency with mat2pm levels
    [n1, n1, nq] = size(q);
    nq = nq - level - 1;
    n1 = n1+1;
    qq = zeros(n1, n1, nq);

    ipm = ipmlevels(level+1) + 1;
    level2 = 2^level;
    for i = 1:nq
        if (i == 1)
            pivot = pm(ipm);
        else
            ipm2 = pmfind(pmidx, pmidx(ipm) - level2);
            pivot = pm(ipm)/pm(ipm2);
        end
        iRight = pmfind(pmidx, pmidx(i + ipmlevels(level+2)) + level2);
        if length(iRight) == 1
            iRight = iRight - ipmlevels(level+2);
            qq(:,:,i) = invschurc(pivot, q(:,:,i), q(:,:,iRight));
        else
            qq(:,:,i) = invschurc(pivot, q(:,:,i), ones(n-level-1));
        end
        ipm = ipm+1;
    end
    q = qq;
end
A = q(:,:,1);
A = deskew(A);

if WARN_A
    % ODF (a) not satisfied
    fprintf(2,...
'PM2MAT: off diagonal zeros found, solution suspect.\n');
end
if WARN_C
    fprintf(2,...
'PM2MAT: multiple solutions to make rank(L-R)=1, solution suspect.\n');
end
```

```
% disable WARN_I for fast version, since it is routinely triggered
%
% Find i0 in pmidx, returning its index, using a binary search,
% since pmidx is in ascending order.
%
% Same functionality as
%
% find(pmidx == i0)
%
% only faster.
%
function i = pmfind(pmidx, i0)
n = length(pmidx);
iLo = 1;
iHi = n;
if pmidx(iHi) <= i0
    if pmidx(iHi) == i0
        i = n;
    else
        i = [];
    end
    return;
end
iOld = -1;
i = iLo;
while i ~= iOld
    iOld = i;
    i = floor((iHi + iLo)/2);
    if pmidx(i) < i0
        iLo = i;
    elseif pmidx(i) > i0
        iHi = i;
    else
        return;
    end
end
i = [];

return;

% invschurc, solveright and deskew are the same as in pm2mat.m
```

Appendix E. V2IDX

```
% V2IDX  Converts a MAT2PM index set (vector) to the index in pm that
% corresponds to a given principal minor.
%
```

```
% For example, if
%
% A = rand(4)
% pm = mat2pm(A)
% idx = v2idx([1 3 4])
%
% then idx = 13 and
%
% det(A([1 3 4],[1 3 4]))
%
% will equal
%
% pm(idx)
%
function idx = v2idx(v)
% The index into pm is simply the binary number with the v(i)'th bit set
% for each i.
n = length(v);          % length of vector containing indices of minor
idx = 0;
for i = 1:n
    idx = idx + bitshift(1,v(i)-1);
end
```

Appendix F. IDX2V

```
% IDX2V  Converts a MAT2PM index into a set of principal minors PM to
% an index set that corresponds to the given principal minor.
%
% For example, if
%
% A = rand(4)
% pm = mat2pm(A)
% v = idx2v(13)
%
% then v = [1 3 4] and
%
% det(A(v,v))
%
% will equal
%
% pm(13)
%
function v = idx2v(idx)
v = [];
i = 1;
while idx ~= 0
```

```
    if bitand(idx, 1) ~= 0
        v = [v i];
    end
    idx = bitshift(idx, -1);    % shift by 1 to the right
    i = i+1;
end
```

Appendix G. PMSHOW

```
% PMSHOW  Displays the given set of principal minors with its index number
% and index sets.
function pmshow(pm)
for i = 1:length(pm)
    v = idx2v(i);
    if imag(pm(i)) == 0
        fprintf(1,'%d\t[%14s]\t%g\n', i, int2str(v), pm(i));
    else % display complex principal minor
        if imag(pm(i)) > 0
            fprintf(1,'%d\t[%14s]\t%g + %gi\n', i, int2str(v),...
                real(pm(i)), imag(pm(i)));
        else
            fprintf(1,'%d\t[%14s]\t%g - %gi\n', i, int2str(v),...
                real(pm(i)), -imag(pm(i)));
        end
    end
end

% IDX2V  Converts a MAT2PM index into a set of principal minors pm to
% an index set that corresponds to the given principal minor.
function v = idx2v(idx)
v = [];
i = 1;
while idx ~= 0
    if bitand(idx, 1) ~= 0
        v = [v i];
    end
    idx = bitshift(idx, -1);    % shift by 1 to the right
    i = i+1;
end
```

References

[1] G.E. Coxson, The P-matrix problem is co-NP-complete, Math. Program. 64 (1994) 173–178.

[2] K. Griffin and M.J. Tsatsomeros, Principal minors, Part I: A method for computing all the principal minors of a matrix, Linear Algebra Appl., in press, doi:10.1016/j.laa.2006.04.008.

[3] O. Holtz, H. Schneider, Open problems on GKK τ-matrices, Linear Algebra Appl. 345 (2002) 263–267.
[4] R. Loewy, Principal minors and diagonal similarity of matrices, Linear Algebra Appl. 78 (1986) 23–64.
[5] E.B. Stouffer, On the independence of principal minors of determinants, Trans. Amer. Math. Soc. 26 (3) (1924) 356–368.